

DESIGN AND ANALYSIS OF PARALLEL ALGORITHMS  
FOR DISTRIBUTED IN-SITU ARRAY BEAMFORMING

By

KEONWOOK KIM

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL  
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT  
OF THE REQUIREMENTS FOR THE DEGREE OF  
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2001

Copyright 2001

by

Keonwook Kim

For Gumhong and Kevin Saeyun...

## ACKNOWLEDGMENTS

I am greatly indebted to Professor Alan George, the chairman of my advisory committee, for his advice, guidance, and inspiration throughout my graduate education. I would like to thank Professor John Harris, Professor Yuguang Fang, and Professor William Hager for serving as members of my supervisory committee. I acknowledge and appreciate the support provided by the Office of Naval Research for this research. I am grateful to the other members of the High-performance Computing and Simulation (HCS) research lab for their encouragement and invaluable help. I also would like to express my thanks to Thomas Phipps from the Applied Research Lab at the University of Texas at Austin for his insight and many useful suggestions.

I thank my parents, parents-in-law, and my brother HyunJai Kim for their love, patience, and support. I also wish to acknowledge all my friends at the University of Florida and elsewhere, especially Sangchoon Kim, Younggoo Kwon, Jeff Markwell, Ryan Forgarty, and Seokjoo Lee. Most of all, I wish to dedicate all my work to my wife Gumhong and my son Kevin Saeyun.

## TABLE OF CONTENTS

	<u>page</u>
ACKNOWLEDGMENTS .....	iv
LIST OF FIGURES .....	vii
ABSTRACT.....	x
 CHAPTERS	
1. INTRODUCTION .....	1
2. BACKGROUND .....	5
2.1. Beamforming Algorithms .....	5
2.2. Parallel and Distributed Computing.....	14
2.2.1. Parallel Computing Model .....	14
2.2.2. Parallel Programming .....	15
2.2.3. Performance Metrics .....	17
2.2.3.1. Execution Time .....	17
2.2.3.2. Speedup.....	17
2.2.3.3. Efficiency.....	18
2.2.3.4. Other Criteria .....	19
3. PARALLEL ALGORITHMS FOR SPLIT-APERTURE CONVENTIONAL BEAMFORMING .....	20
3.1. SA-CBF Algorithm .....	20
3.2. Analysis of the SA-CBF .....	26
3.3. Parallel Algorithms for the SA-CBF .....	29
3.3.1. Iteration Decomposition.....	31
3.3.2. Angle Decomposition .....	33
3.4. Performance Analysis of Parallel SA-CBF Algorithms .....	36
3.4.1. Sequential Execution Time .....	36
3.4.2. Parallel Execution Time.....	37
3.4.3. Scaled Speedup and Parallel Efficiency .....	40
3.4.4. Data Memory Capacity .....	42
3.5. Summary.....	43

4. PARALLEL ALGORITHMS FOR SUBSPACE PROJECTION BEAMFORMING	44
4.1. SPB Algorithm	46
4.2. Analysis of the SPB	52
4.3. Parallel Algorithms for the SPB	55
4.3.1. Iteration Decomposition	55
4.3.2. Frequency Decomposition	57
4.4. Performance Analysis of Parallel SPB Algorithms	59
4.4.1. Sequential Execution Time	59
4.4.2. Parallel Execution Time	60
4.4.3. Computation vs. Communication	62
4.4.4. Scaled Speedup and Parallel Efficiency	64
4.4.5. Result Latency	65
4.4.6. Data Memory Capacity	66
4.5. Summary	67
5. PARALLEL ALGORITHMS FOR CONVENTIONAL MATCHED-FIELD PROCESSING	69
5.1. CMFP Algorithm	74
5.2. Analysis of the CMFP	79
5.3. Parallel Algorithms for the CMFP	83
5.3.1. Frequency Decomposition	83
5.3.2. Section Decomposition	85
5.4. Experimental Testbeds	87
5.4.1. PC Cluster	88
5.4.2. DSP Array	88
5.5. Performance Analysis of Parallel CMFP Algorithms	90
5.5.1. Sequential Execution Time	91
5.5.2. Parallel Execution Time	92
5.5.3. Communication Time	95
5.5.4. Scaled Speedup and Parallel Efficiency	97
5.5.5. Data Memory Capacity	99
5.5.6. Power and Energy Consumption	100
5.6. Summary	105
6. CONCLUSIONS	107
LIST OF REFERENCES	111
BIOGRAPHICAL SKETCH	116

## LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
1.1. Projected increase in the computational requirements for sonar beamforming.....	2
2.1. Conventional beamformer .....	7
2.2. Simple illustration of steering effect (ideal case).....	8
2.3. Normalized beam patterns for an 8-node configuration .....	9
2.4. Parallel computing architecture model .....	15
3.1. Geometry of the Split-Aperture Conventional Beamformer.....	21
3.2. Sequential SA-CBF algorithm for 8 nodes .....	24
3.3. SA-CBF characteristic .....	25
3.4. Pseudo-code for the sequential SA-CBF algorithm.....	26
3.5. Average execution time and memory requirement for MM and MC model .....	28
3.6. Averaged execution time comparison for SA-CBF in an 8-node configuration.....	29
3.7. Block diagram of the iteration decomposition in a 3-node configuration .....	32
3.8. Pseudo-code for the iteration decomposition.....	33
3.9. Block diagram of the angle decomposition in a 3-node configuration.....	34
3.10. Pseudo-code for the angle decomposition .....	35
3.11. Average sequential execution time per iteration.....	37
3.12. Average parallel execution time per iteration.....	38
3.13. Upshot profiles for both of the parallel SA-CBF algorithms with 4 nodes .....	40
3.14. Performance of parallel SA-CBF algorithms.....	41

3.15.	Memory requirement of the steering stage .....	42
4.1.	Sequential SPB algorithm .....	50
4.2.	The required number of multiplication operations .....	51
4.3.	Number of multiplications for each stage .....	54
4.4.	Block diagram of the iteration decomposition in a 3-node configuration .....	55
4.5.	Block diagram of the frequency decomposition in a 3-node configuration .....	58
4.6.	Average sequential execution time per iteration .....	60
4.7.	Average parallel execution time .....	61
4.8.	Communication of the parallel SPB algorithms .....	63
4.9.	Performance of parallel SPB algorithms .....	64
4.10.	Average result latency time per beamforming job .....	66
4.11.	Memory requirement of the steering stage as a function of nodes .....	67
5.1.	Normal-mode propagation with a source at 1000m depth .....	76
5.2.	CMFP block diagram .....	77
5.3.	Sample output of the CMFP for a 32-node array .....	78
5.4.	Computational procedure of the KRAKEN normal-mode solution .....	80
5.5.	Frequency decomposition .....	84
5.6.	Section decomposition .....	86
5.7.	Digital signal processor array and its node architecture .....	89
5.8.	Sequential execution times vs. number of sensor nodes .....	91
5.9.	Parallel execution time of frequency decomposition vs. number of nodes .....	93
5.10.	Parallel execution time of section decomposition vs. number of nodes .....	94
5.11.	Communication with parallel algorithms .....	97
5.12.	Parallel performance .....	98
5.13.	Data memory requirement per node for the steering stage .....	100



5.14.	Power consumption of CMFP in the DSP array .....	103
5.15.	Energy consumption of CMFP in the DSP array.....	104
5.16.	Energy distribution efficiency of parallel CMFP.....	105

Abstract of Dissertation Presented to the Graduate School  
of the University of Florida in Partial Fulfillment of the  
Requirements for the Degree of Doctor of Philosophy

DESIGN AND ANALYSIS OF PARALLEL ALGORITHMS  
FOR DISTRIBUTED IN-SITU ARRAY BEAMFORMING

By

Keonwook Kim

August 2001

Chairman: Dr. Alan D. George

Major Department: Electrical and Computer Engineering

Parallel processing algorithms, coupled with advanced networking and distributed computing architectures, improve the overall computational performance, dependability, and versatility of a digital signal processing system. In this dissertation, several novel parallel algorithms are introduced and investigated for beamforming algorithms that include split-aperture conventional beamforming (SA-CBF), subspace projection beamforming (SPB), and conventional matched-field processing (CMFP) algorithms. Each beamforming algorithm has its distinctive complexity and bottlenecks caused by the unique computational pattern of the algorithm. Based on a specific domain, each parallel algorithm decomposes the sequential workload in order to obtain scalable parallel speedup and efficient memory management. Depending on the processing requirement of the beamforming algorithm, the computational performance of the parallel algorithm reveals different characteristics. The least computationally complex algorithm, SA-CBF, is particularly susceptible to the communication overhead. With parallel SA-CBF

algorithms, the best combination is shown to yield around 80% scaled efficiency on a cluster of workstations. The two high-complexity algorithms, SPB and CMFP, show scalable parallel performance on the testbed. The impact on parallel performance due to workload balancing, communication scheme, algorithm complexity, processor speed, network performance, and testbed configuration is explored. Finally, results show almost ideal distributed energy characteristics over the multiple processors used by the parallel algorithms in CMFP.

## CHAPTER 1 INTRODUCTION

Beamforming is a class of array processing algorithms that optimizes an array gain in a direction of interest and/or locates directions of arrival (DOA) for sources. In sonar, radar, and recently wireless communication systems, the beamforming algorithms are particularly vital applications. Over the past several decades, rapid advancements in the mathematical algorithms and translation techniques have resulted in high-performance beamforming algorithms. Profound understanding of signal and noise has produced sophisticated algorithms to perform beamforming in a high clutter environment. Continuous innovations in beamforming, such as adaptive and matched-field processing, lead to beamforming algorithms that are better able to cope with attenuated signal and noisy environments.

During the last ten to fifteen years, the modern high-speed digital processor has also improved implementation techniques for beamforming algorithms. Consequently, there have been obvious transitions from analog processing to digital processing and from manual hardware-based to automated software-based systems. Recently, these trends have been furthered by the tremendous increase in processing capability that allows computationally intensive techniques to be implemented. However, still the sequential implementation of the beamforming algorithms for many sensors and over a wide band of frequencies often requires intensive computation that may exceed the pace of conventional processor performance. In order to implement beamforming algorithms for array signal processing in real-time, considerable processing power is necessary to cope

with these demands. Figure 1.1 shows the projected increase in the computational complexity of present and next-generation sonar beamformers [BER96,LIU98].

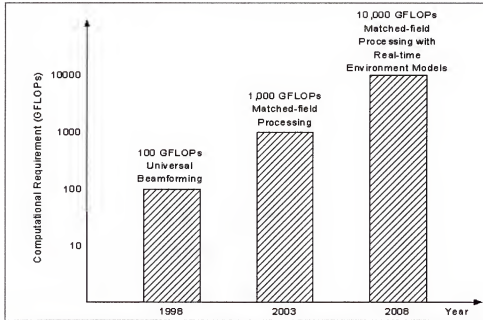


Figure 1.1. Projected increase in the computational requirements for sonar beamforming

The use of parallel and distributed computing to perform the computationally intensive beamforming algorithms represents an alternative to a sheer increase in processor speed. Parallel processing algorithms coupled with advanced networking and distributed computing architectures can be used to turn the telemetry nodes of arrays into processing nodes and thereby perform as a distributed processing system for autonomous, in-situ beamforming. A distributed processing approach holds the potential to eliminate the need for a centralized data collector and processor, and increase overall computational performance, dependability, and versatility.

The parallel algorithms considered in this dissertation are designed for such a distributed system. In this parallel model, an augmentation in the number of nodes will increase processing power and the problem size as well. Such an architecture ties the

degree of parallelism (DOP) to the number of physical nodes in the target system. By careful decomposition and mapping of the sequential workload over multiple processors, a speedup in the execution time is expected for the parallel algorithms. Numerous decomposition methods exist for each beamforming algorithm in terms of iteration, angle, frequency, and other domains.

In the first part of this dissertation, novel parallel algorithms for a split-aperture conventional beamforming (SA-CBF) algorithm will be presented and analyzed. The SA-CBF algorithm is a modified version of the conventional beamforming (CBF) algorithm to improve computational performance by cross-correlation between two sub-arrays. By decomposing the workload, the iteration decomposition and angle decomposition algorithms enhance computational performance of the SA-CBF as well as the memory requirement of the parallel algorithm.

Compared to the CBF and SA-CBF algorithms, the adaptive beamforming (ABF) algorithms improve beamforming performance at the expense of computational increase. This computational rise is usually expressed as the form of additional manipulation stages in the beamforming algorithm. The changes in the computational flow of the beamforming algorithm affect the development of parallel algorithms. One of the ABF algorithms, subspace projection beamforming (SPB), will be introduced and exploited. Based on the SPB algorithm, unique parallel algorithms will be devised and investigated in the second phase of this dissertation

The third phase of this research consists of the development of parallel algorithms for the conventional match-field processing (CMFP) algorithm. The CMFP algorithm is an advanced algorithm established from the signal propagation model in an ocean

waveguide. To generate steering vectors, the tremendous computation and memory requirements in the initial phase are an inevitable property of the CMFP algorithm. As a result, an in-depth discussion about the parallel algorithms proposed to ascertain the difficulty of CMFP implementation is a critical part of the last phase.

The remainder of this dissertation is organized as follows. In Chapter 2, background on basic beamforming algorithms and parallel distributed computing is provided. In Chapter 3 through Chapter 5, novel parallel algorithms for SA-CBF, SPB, and CMFP, respectively, are presented and evaluated. Chapter 6 concludes with a summary of the dissertation research.

## CHAPTER 2 BACKGROUND

In this chapter, the basic concept of beamforming is explained in detail to establish a foundation for understanding more advanced beamforming algorithms. Brief descriptions of the SA-CBF, ABF, and MFP beamforming algorithms are also given. Finally, an overview of parallel and distributed computing concepts for parallel beamforming is presented.

### 2.1. Beamforming Algorithms

As previously mentioned, beamforming is a class of array processing that optimizes an array gain in a direction of interest and/or locates directions of arrival (DOA) for sources. The determination of the DOA relies on the detection of the time delay of the signal between sensors. Incoming signals are steered by complex-number vectors in the frequency domain. If the beamformer is properly steered to an incoming signal, the multi-channel-input signals will be amplified coherently, maximizing power in the beamformed output; otherwise, the output of the beamformer is attenuated to some degree. Thus, peak points in the beamforming output indicate DOA for sources.

Beamforming is based on application of a delay-and-sum algorithm in either the frequency or time domain. In general, there are three important assumptions used in the beamforming algorithms. First, signals arriving at the sensors from a source are highly correlated with an acceptable signal-to-noise ratio (SNR). Second, ambient noise in the environment is uncorrelated from sensor to sensor. Third, sources are located far enough from the array so that their signals may be estimated as plane waves, and the DOA is



approximately equal at all sensors. With these assumptions, the wavefront of a source signal arrives at each sensor with a simple time delay. The beamforming algorithm detects time delays of the signal between sensors. Time information and configuration of the sensor array are used to determine the DOA for the signal source.

The digital beamformer obtains a data set from each sensor by sampling an analog signal. Each data set may be partially overlapped with the previous set, contributing to better spectral stability. Furthermore, a time-window function may be applied to the data, aiding the spectral estimation. The location of the source is unknown to the array, but the DOA can be found by steering the data for every possible angle. In order to steer data to a specific look direction, individual data vectors from each sensor are digitally delayed or advanced by some time, which is a function of the look direction. The time shift in the time domain can be described as a multiplication with some complex exponential value in the frequency domain, as shown in Equation 2.1.

$$x[n - n_d] \xleftrightarrow{\text{Fourier Transform}} e^{j\omega n_d} X(e^{j\omega}) \quad (2.1)$$

In this equation,  $x[n]$  is the time sequence,  $n$  is the discrete time index,  $n_d$  is delay of the system, and  $X(e^{j\omega})$  is the Fourier transform of  $x[n]$  as a function of angular frequency  $\omega$ . The amount of time shift for each sensor is calculated using Equation 2.2 with the configuration shown in Figure 2.1a. In the equation below,  $r_k$  is the position of node  $k$ ,  $C_{\text{sound}}$  is the speed of sound in water,  $c$  is the position of phase center which is the reference point for the beamforming, and  $\theta$  is the steering angle of the array.

$$\tau_k = \frac{\sin \theta \cdot (r_k - c)}{C_{\text{sound}}} = \frac{d}{C_{\text{sound}}} \quad (2.2)$$

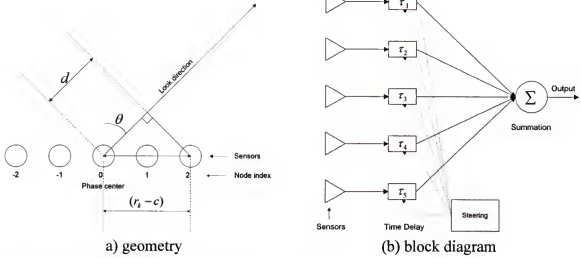


Figure 2.1. Conventional beamformer

The plane wave steering vector (i.e. the exponential factor in Equation 2.1) multiplies the input Fourier-transformed data; hence, the input data of each node are steered to the specific direction in relation to the phase center. These steered vectors are added sample by sample to the output, as illustrated in Figure 2.1b. With proper steering, the input vector signals will be magnified coherently; otherwise, the output of the beamformer is attenuated. Figure 2.2 demonstrates the effects of steering with a simple 2-node example. When the steering angle is incorrect for the incoming signal (Figure 2.2a and Figure 2.2b), the resulting beamformer output power is decreased from its maximum (Figure 2.2c). When the steering angle is such that the time difference experienced between the two nodes is neutralized (Figure 2.2d and Figure 2.2e), the power of the output signal is maximized (Figure 2.2f). The final beamformer output is obtained by plotting the signal power of each steering angle. The peak point of the plot indicates the DOA of an acoustic source.

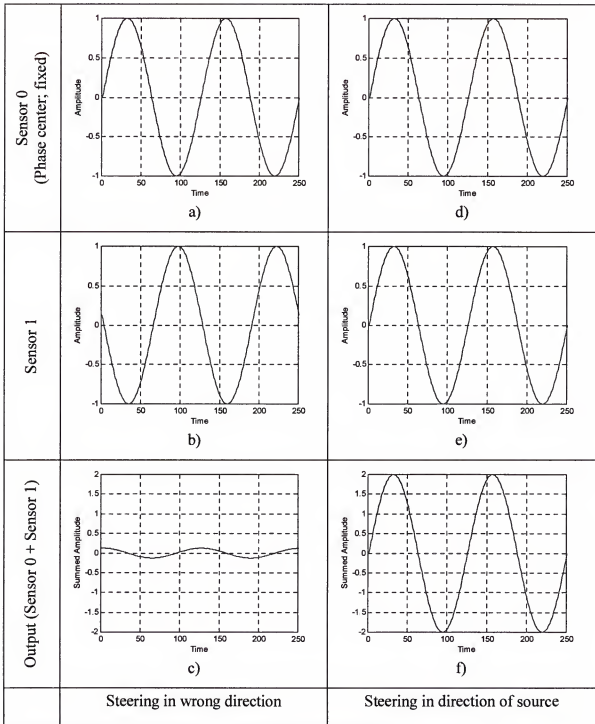


Figure 2.2. Simple illustration of steering effect (ideal case)

For  $N$  sensors of a beamformer, plane-wave steering vectors are defined as in Equation 2.3 for a uniformly spaced linear array configuration.

$$s(\theta) = \left[ 1 \quad e^{-j2\pi f \frac{D \sin \theta}{c}} \quad e^{-j2\pi f \frac{2D \sin \theta}{c}} \quad e^{-j2\pi f \frac{3D \sin \theta}{c}} \dots e^{-j2\pi f \frac{(N-1)D \sin \theta}{c}} \right]^T \quad (2.3)$$

In this equation,  $f$  is processing frequency,  $D$  is distance between sensors,  $c$  is speed of sound, and  $\theta$  is steering direction. The processing frequency should be carefully chosen for a beamforming algorithm. Above a certain frequency, spatial aliasing (which will make targets appear in more than one location) is expected, and below that frequency, degradation of angular resolution (broadening of target peaks) is anticipated. Figure 2.3 shows that at lower frequencies the main lobe of the beam pattern tends to be wider, leading to degradation of angular resolution, and at higher frequencies the spatial aliasing is clearly presented.

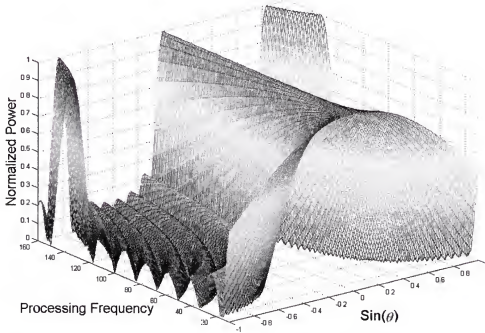


Figure 2.3. Normalized beam patterns for an 8-node configuration

The beamformer samples data for a finite length of space and time, and a limited number of sensors. To improve the spectral characteristics of data, it is common to introduce shading windows on the finite length signal over time. This weight, such as a Hamming window, attempts to enhance the peaks and the nulls of the data spectrum over that for the rectangular window with no weight. Nuttall [NUT81] suggested optimal behavior windows under several different constraints. These windows can be applied over sensors for spatial weighting as well. The main beam width and side lobe levels are controlled by the spatial window and node configuration.

The complex scalar output of the beamformer is expressed as a steered summation of the received data as shown in Equation 2.4:

$$y(t) = s(\theta)^H \cdot x(t) \quad (2.4)$$

where  $s(\theta)$  is the steering vector defined in Equation 2.3,  $x(t)$  is the vector of the Fourier component of received data near the processing frequency, and operator  $H$  represents complex conjugation and matrix transposition. The output power spectral density of the beamformer is then given by

$$\begin{aligned} P(\theta) &= E\left[y(t)^2\right] = E\left[s(\theta)^H \cdot x(t) \cdot x(t)^H \cdot s(\theta)\right] = s(\theta)^H \cdot E\left[x(t)^H \cdot x(t)\right] \cdot s(\theta) \\ &= s(\theta)^H \cdot R \cdot s(\theta) \end{aligned} \quad (2.5)$$

where  $R$  is the cross-spectral matrix (CSM).

In the fundamental beamforming algorithm previously described, known as conventional beamforming (CBF), the most computationally intensive part is the multiplication between the CSM and the steering vectors. The CBF algorithm applies a proper delay on each node data to look at a specific direction and compute signal power for the corresponding direction. Since the DOAs depend on the beamforming output power, the algorithm needs to estimate signal powers in all potential directions in order to

locate DOA of sources. The computational requirement of the steering stage is significantly increased in term of number of nodes, number of steering angles, and number of frequency bins as shown in Equation 2.5. To overcome the computational problem of CBF, the split-aperture conventional beamforming (SA-CBF) algorithm introduces an interpolation method based on the cross-correlation between two sub-arrays. The SA-CBF array is logically divided into two sub-arrays. Each sub-array independently performs CBF using steering vectors on its own data. The two sub-array CBF outputs are cross-correlated to detect the time delay of the signal for each interpolated steering angle. The cross-correlated data, with knowledge of the steering angles and several other parameters, will map the final SA-CBF output. Unlike the CBF algorithm, the SA-CBF algorithm does not need to steer at every individual desired angle in the steering stage. By cross-correlating to time delays slightly offset from the sub-array steering delays, each sub-array steering angle can be used to generate a range of the time delay plot. Compared with CBF computation, the SA-CBF significantly reduces computation by the interpolation. Further discussion on sequential and parallel SA-CBF algorithms is provided in Chapter 3.

In general, the class of CBF algorithms, which includes SA-CBF, shows inferior beamforming performance in most situations due to the vulnerability of the algorithm to incoming noise. Frequently, the CBF output at a steered direction can be contaminated by the unwanted signals, which come from directions other than look directions. Since the level of the side lobes and the width of the main lobe are proportional to the number of nodes, the CBF generates a very wide and smooth output with a limited number of nodes. A large number of nodes is required for CBF to resolve DOA with precise accuracy.

Adaptive beamforming (ABF) techniques improve the performance of the beamformer by maximizing signal and minimizing noise power of the output for a steered direction. With certain constraints, an ABF algorithm optimizes the cost function for better beamforming output. The spatial weight from the cost function relies on incoming data type; therefore, the weight vector is computed and optimized based on the data model and beamformer configuration. These improvements generally require extra manipulations to compute data-dependent weight vectors, and there are numerous such operations depending upon what kind of ABF we apply. Further information on ABF algorithms is given in Chapter 4.

Recent array processing techniques have exploited the physics of wave propagation in order to improve the detection capabilities of arrays for beamforming. By exploiting environmental parameters, it is possible to obtain improved detection performance. Matched-field processing (MFP) is one such method considering the environmental complexities of the acoustics field in an ocean waveguide. MFP works by matching a received acoustic data vector to a set of steering vectors produced by modeling the environment. This data model considers the complex mode/multipath structure of the ocean waveguide; hence, the increased complexity of the data model provides more information about source location such as range and depth. This approach is in contrast to conventional techniques based on a plane-wave model, which only take into account the single direct path of wave propagation. Due to the data model limitation, the plane-wave beamformer is simply able to resolve angle information of the source. Conversely, the complex data model of the MFP algorithm may trigger imprecise estimation of source location by insufficient and/or inaccurate information about the environment. Additional background on the MFP algorithm is presented in Chapter 5.

Throughout this work, the wavefield is assumed to be generated by a finite number of targets, and contain information about signal parameters characterizing the source in this experiment. For an isotropic and point source, the positions lying on the surface of the sphere of radius will then share a common phase and are referred to as a wavefront. In the far-field case, the radius of the wavefront is so large that a flat plane of constant phase can be considered, thus resulting in a plane wave. The plane waves arrive at each sensor at different times based on the array configuration and physical parameters. In the frequency domain, sampled data from each sensor are modeled as a linear combination of the separate signal vectors plus ambient noise as shown below.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix} = \begin{bmatrix} a(\theta_1) & a(\theta_2) & \cdots & a(\theta_D) \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_D \end{bmatrix} + \begin{bmatrix} W_1 \\ W_2 \\ \vdots \\ W_M \end{bmatrix} \quad (2.6)$$

$$x = AF + W$$

In this equation,  $a(\theta_j)$  is the mode vector responding to the DOA  $\theta_j$  of the  $j$ th signal,  $F_j$  is the  $j$ th incident signal,  $W$  is the noise vector (isotropic noise used),  $D$  is the number of sources, and  $M$  is the number of the nodes. Note that vector  $x$  consists of one frequency bin element from each node.

This data model is the ideal plane-wave model for general beamforming algorithms such as CBF, SA-CBF, ABF, etc. In reality, because the ocean can be regarded as a horizontal waveguide for the transmission of acoustic signals and noise, the simple plane-wave model is insufficient to describe the complex wave propagation. The MFP algorithm requires and exploits a realistic data model based on the complex wave equation. An explanation of the MFP steering vector and data is presented in Chapter 5.



## 2.2. Parallel and Distributed Computing

Parallel and distributed computing (PDC) is a very broad and deep field of study, and it is important to gain a basic understanding of the background of PDC before examining the approach taken in this research. The term ‘parallel’ signifies that the decomposition of the beamforming workload is mapped across many processors. Distributed means that the processors communicate via message passing, as opposed to shared memory. Further discussion about PDC is presented in the following sections. Where applicable, the issue of how the beamforming array relates to the scheme of parallel and distributed computing is addressed.

### 2.2.1. Parallel Computing Model

In this research, a basic sonar array is assumed to contain an arbitrary number of single transducer elements strung together in a linear fashion. In the baseline sequential model, the dumb nodes do not contain a microprocessor but do contain some electronics for the network protocol and an analog-to-digital converter to sample the data from the transducer. With that configuration, the dumb node has only the capability of sending gathered data to the centralized data collector/processor for computing the beamforming output. However, the architecture of a smart node, which is used for parallel computing, is assumed to have a single transducer with processing capability. The basic layout of such a parallel architecture is shown below in Figure 2.4.

Each node in the parallel processing system has one main processor, a memory device, an A/D converter, a controller for network communication, a battery, and other various interface/power components. The parallel computing model used in this research consists of loosely coupled smart nodes connected by a network. Each node incorporates a processor with its own local memory. Because the memory is not shared, the

processors must communicate through message passing. A processor, along with its memory, essentially forms a stand-alone computer. This linear parallel beamforming array model proposes an array of such computers and may thus be classified as a multicomputer system. Multicomputers are desirable in a multitude of situations because they provide flexibility, scalability, and cost effectiveness.

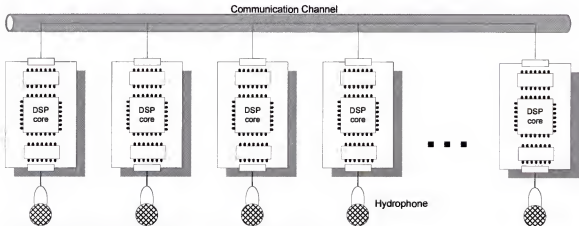


Figure 2.4. Parallel computing architecture model

### 2.2.2. Parallel Programming

In an ideal environment, the programmer simply feeds the compiler a sequential program from which it generates the parallel program implicitly. The compiler determines all of the inherent algorithmic parallelism and communication necessary to execute the program in parallel. This implicit approach has been implemented in a limited number of languages, but it is only remotely as efficient as explicit parallelization. In languages using the explicit approach, the programmer sets off the instructions to be executed in parallel by adding parallel constructs to existing languages such as C or FORTRAN.

The programming model used in this research is message passing on a cluster of computers by way of the message-passing interface (MPI) [MPI94], which is an adopted

standard that defines the syntax of a core set of parallel communication functions. This explicit approach to deriving parallel programs, such as constructing parallel beamforming programs, allows the programmer to determine exactly where and how much communication will take place by adding parallel constructs to C- or FORTRAN-based programs. This capability is important in the design of new parallel programs because it gives the user the flexibility required in order to discover the most efficient parallelization techniques. In addition, the beamforming array is likely to be distributed in nature and thus benefit from a message-passing environment. The explicit approach over a cluster of computers efficiently supports the coarse- and medium-grained decomposition of beamforming algorithms, while the fine-grained decomposition is not recommended in this parallel model due to the significant interprocessor communication overhead.

In parallel programming via MPI, each processor is assigned a special number called rank and its task is determined by this rank number. There is one program where the programmer must dictate the parallelism explicitly, and every processor executes it. The processors may take different paths through the program because they evaluate conditional statements differently; however, every processor is executing the same program [PAC95].

As with any parallel program, dependencies of the algorithm are carefully examined and considered in the decomposition of the program. The sequential workload of beamforming is distributed over the processor based on the dependency analysis. The following chapters present an overview of the proposed parallel decomposition techniques with different levels of granularity via iteration decomposition, angle decomposition, and other decomposition methods.

### 2.2.3. Performance Metrics

In order to measure the effectiveness of the parallelization of a program, certain concrete performance metrics must be introduced and examined. Although the evaluation of the overall quality of the program is both objective and subjective, the following terms are ways of objectively measuring the performance of a parallel program.

#### 2.2.3.1. Execution Time

The execution time is defined as the time elapsed from the beginning of the program execution to the end of program execution and is an important factor in determining the responsiveness of the system. If an algorithm is to be executed in real-time, as will the processing in a beamforming array, then the execution time is crucial to the success of the algorithm. The three main components of execution time include computation time, communication time, and idle time. Upshot, a profiling tool used with MPI, will be employed to illustrate the times spent in each of these components graphically [SNI96].

#### 2.2.3.2. Speedup

Speedup describes how much faster a parallel algorithm executes as opposed to the sequential algorithm. Speedup is defined as [ZOM96]:

$$S = \frac{T_s}{T_p} \quad (2.7)$$

where  $T_s$  is running time of best available sequential algorithm and  $T_p$  is running time of the parallel algorithm on a collection of processors

The larger the speedup, the better the quality of the algorithm. Two rules that are commonly referred to in parallel computing are Amdahl's law and Gustafson's law.

Amdahl's law limits the speedup to the amount of inherent parallelism found in the algorithm [AMD67]. Amdahl's law states the following:

$$S_N = \frac{T_s}{T_p} = \frac{T_s}{\alpha T_s + \frac{(1-\alpha)T_s}{N}} \quad (2.8)$$

where  $S_N$  is speedup for an  $N$ -processor system and  $\alpha$  is fraction of algorithm that is not parallelizable (i.e. Amdahl's fraction)

The alternative to Amdahl's law is Gustafson's law. Gene Gustafson recognized that Amdahl's law is valid only for a fixed workload size and is thus not scalable [GUS88]. For the case where the problem size scales with the number of processors (i.e. where  $\alpha$  is a function of the problem size), Gustafson derived a speedup based on a fixed time concept that led to a scaled speedup model [HWA93]:

$$S'_N = \frac{N}{1 + (N-1)\alpha(N)} \quad (2.9)$$

where  $S'_N$  is a scaled speedup for an  $N$ -processor system and  $\alpha(N)$  is the fraction of the algorithm that is not parallelizable

In many cases, speedup is the ultimate goal when parallelizing a sequential program. However, in the case of the beamforming array, speedup is only one of the many reasons why the beamforming algorithms are partitioned, decomposed, and mapped into parallel forms. Other reasons may include reliability, survivability, energy consumption per node, weight per node, etc.

#### 2.2.3.3. Efficiency

The efficiency is defined as the speedup divided by the number of processors used. The efficiency is highest when all processors are used throughout the entire execution of the program, and lowest efficiency occurs when the program runs almost

exclusively on a single processor or the communication overhead dwarfs the computation time of each process.

#### 2.2.3.4. Other Criteria

In addition to the criteria listed above, the performance of the parallel beamformer can be measured by various other aspects, such as memory requirements, result latency, throughput, power/energy consumption, etc. All of these metrics are important to the beamforming array application to analyze feasibility and real-time requirements.

The proposed beamforming array must be a multicomputer with distributed processing for several reasons. First of all, the array will not support communication links fast enough to merit shared memory. Secondly, the spatially distributed nature of the array will require that each sensor have its own processor and be able to perform calculations on the data independently of what is happening on the other nodes. In order to reach data at other nodes, the information must be explicitly passed to the next node, which is a message-passing communication technique. Thus, parallel beamforming experiments are performed over a cluster of computers with an interconnection network.

Although this chapter only briefly touches upon aspects of parallel computing, enough information is provided so that techniques used to parallelize the beamforming array can be understood and its niche identified in the grand scheme of parallel and distributed computing.

### CHAPTER 3

#### PARALLEL ALGORITHMS FOR SPLIT-APERTURE CONVENTIONAL BEAMFORMING

In this chapter, novel decomposition techniques for SA-CBF are proposed and analyzed. A theoretical background of SA-CBF in viewpoints of mathematical and computational aspects is first described; then, two unique decomposition techniques are designed for SA-CBF. The parallel experiments are performed on a cluster of computers for performance analyses in terms of speedup, parallel efficiency, and memory requirements. Finally, conclusions about the decomposition techniques are presented in the last section. This chapter is published material [GEO99] and served as the first phase of this research.

#### 3.1. SA-CBF Algorithm

SA-CBF is based on single-aperture conventional beamforming in the frequency domain. The beamforming array is logically divided into two sub-arrays, shown in Figure 3.1 [GEO98,MAC90,STE91]. Each sub-array independently performs conventional frequency-domain beamforming using steering vectors on its own data. The two sub-array beamforming outputs are cross-correlated to detect the time delay of the signal for each steering angle. The cross-correlated data, with knowledge of the steering angles and several other parameters, will map the final beamforming output.

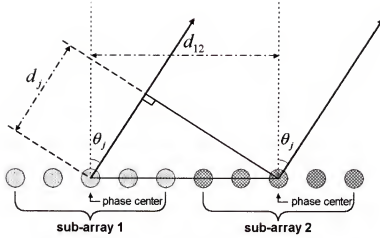


Figure 3.1. Geometry of the Split-Aperture Conventional Beamformer

Unlike the single-aperture beamforming algorithm (i.e. CBF), the SA-CBF algorithm does not need to steer at every individual desired angle in the steering stage. The cross-correlation creates some redundant information between the adjacent sub-array steering angles. By cross-correlating to time delays slightly offset from the sub-array steering delays, each sub-array steering angle can be used to generate a range of the time delay plot. The discrete cross-correlation function is defined in Equation 3.1:

$$c_{xy}(n) = \frac{1}{L} \sum_{i=0}^{L-1} x(i)y(i+n) \quad \xleftrightarrow{\text{Fourier Transform}} \quad C_{xy}(k) = X(k)Y(k)^* \quad (3.1)$$

$$n = 0, 1, \dots, L-1 \quad k = 0, 1, \dots, 2L-2$$

where vectors  $x$  and  $y$  are the sub-array beamforming outputs,  $L$  is the number of samples in  $x$  and  $y$ , and operator  $*$  indicates complex conjugation. We are only interested in the small number of angles or time delays in the cross-correlation adjacent to the beamforming angle of the sub-array.

Before the inverse Fourier transform is applied to obtain the cross-correlation as a function of time delay, the smoothed coherent transform (SCOT) is used to prefilter the



cross-correlation. The spectral whitening obtained by SCOT results in an improved signal-to-interference ratio and an enhancement of the correlation between the two sub-arrays. Additional advantages of SCOT, as well as other prefiltering techniques, can be found in Ferguson [FER89]. The SCOT weighting function is given by Equation 3.2.

$$|W(k)|^2 = \frac{1}{\sqrt{|X(k)|^2 |Y(k)|^2}} \quad (3.2)$$

SCOT is accomplished either by taking the instantaneous magnitude of the cross-correlation in frequency or a running average of the magnitude.

As mentioned previously, each beam formed by the sub-arrays can be used to calculate multiple points in plotting the output bearing. The process by which this increase in resolution is accomplished is called t-interpolation. For each output angle,  $\tau$ -interpolation works with only the two cross-correlation results that are nearest to the desired angle. Raised-cosine weights are used to calculate the beamforming output for the interpolated angle from a linear combination of the two adjacent cross-correlation values. Because we need only a limited range of cross-correlation values to calculate the final beamforming output, the inverse discrete Fourier transform is then performed using Equation 3.3 rather than the conventional FFT algorithm. This method will decrease computational cost of the inverse Fourier transform stage.

$$c(\tau_j) = \frac{1}{N_{FFT}} \sum_{m=M_1}^{M_2} 2 \operatorname{Re} \left[ \tilde{C}_m e^{j2\pi(m-1)\Delta f \tau_j} \right] \quad (3.3)$$

In Equation 3.3,  $M_1$  and  $M_2$  are the minimum and maximum frequency bin numbers in which we are interested,  $\tilde{C}_m$  is the weighted and normalized frequency-domain cross-correlation,  $\Delta f$  is the frequency resolution of the FFT,  $\tau_j$  is the time delay

between phase centers, and  $Re(\cdot)$  is the function that returns the real value of a complex number.

The two nearest cross-correlation vectors, subscripted as  $c_L(\tau_j)$  and  $c_R(\tau_j)$ , are used to evaluate each interpolated output angle via  $\tau$ -interpolation as defined in Equation 3.4 and Equation 3.5.

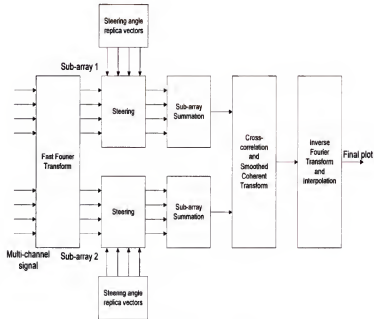
$$c(\theta_{out}) = h_L c_L(\tau_j) + h_R c_R(\tau_j) \quad (3.4)$$

$$h_L = \frac{1}{2} \left[ 1 + \cos \pi \left( \frac{\theta_L - \theta_{out}}{\theta_L - \theta_R} \right) \right] ; \quad h_R = 1 - h_L \quad (3.5)$$

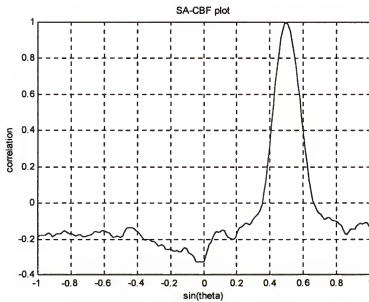
In these equations,  $\theta_L$  and  $\theta_R$  are the sub-array beamformed angles to the left and right, respectively, of the output interpolated angle  $\theta_{out}$ , and  $c_L(\tau_j)$  and  $c_R(\tau_j)$  are cross-correlation values for angles  $\theta_L$  and  $\theta_R$ , respectively. The final output of the SA-CBF beamformer finds  $c(\theta_{out})$  versus each interpolated  $\theta_{out}$ . Further information on the design of the sequential SA-CBF algorithm can be found in Machell [MAC90] and the experimental evaluation of SA-CBF can be found in Stergiopoulos and Ashley [STE91]. To consolidate the above processes, Figure 3.2a shows the block diagram of the SA-CBF algorithm, and Figure 3.2b is a sample output of the SA-CBF.

It is often highly desirable to increase the number of input nodes and the number of steering angles in a beamforming system. By increasing the number of input sensors, the beamformer resolves finer angles because the main lobe of the beam pattern narrows and the side lobes of the beam pattern decrease. Figure 3.3a, which is constructed using a polynomial representation of a linear array [DAV67], shows this result. Even if there are enough nodes to obtain a sharp beam pattern, the number of steering angles remains an important factor in producing high-resolution beamforming output. The number of

steering angles decides the number of output points of the beamformer; thus, a small number of steering angles may cause a blurry beamforming output.

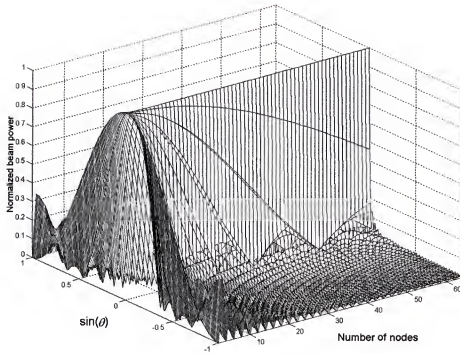


a) block diagram

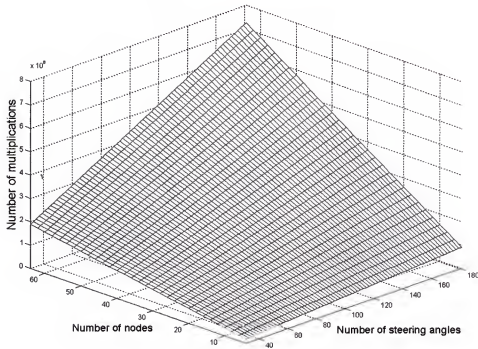


b) sample output with source at  $\theta=30^\circ$

Figure 3.2. Sequential SA-CBF algorithm for 8 nodes



a) normalized beam patterns



b) number of multiplication requirement

Figure 3.3. SA-CBF characteristic

As both of these parameters increase, the number of multiplication operations required to generate beamforming output is increased rapidly, as shown in Figure 3.3b. According to this figure, powerful processing is essential to generate high-resolution beamforming output with acceptable latency and throughput. In cases where current technology cannot provide sufficient real-time performance in a single processor, a trade-off will be required between response time and resolution. The scalable performance of parallel processing will help to overcome limits imposed by a single front-end processor.

### 3.2. Analysis of the SA-CBF

The SA-CBF algorithm previously discussed can be split into five distinct stages: FFT, Steering, Sub-array Summation, Cross-correlation/SCOT, and Inverse Fourier Transform/Interpolation. High-level pseudo-code for SA-CBF is shown in Figure 3.4. Up to and including the sub-array summation, SA-CBF is a frequency-domain conventional beamforming algorithm except that there are two phase centers. The succeeding stages improve the visual resolution and bearing estimation of the beamformer output using digital signal processing techniques.

```

Do FFT for every incoming signal vector;
For j=1, number of steering angles
    Steering(j);
    Sub-array summation(j);
    Cross-correlation and SCOT(j);
End
For k=1, number of output angles
    Inverse Fourier transform and interpolation(k);
End

```

Figure 3.4. Pseudo-code for the sequential SA-CBF algorithm

In building a baseline for the parallel SA-CBF algorithms, two different sequential implementations were created, one using a minimum-memory (MM) model and the other using a minimum-calculation (MC) model. The MM model tries to save memory by doing more calculations, and the MC model saves redundant calculations by using more memory. To illustrate the trade-off involved in selecting one model over the other, consider the steering vectors and the inverse Fourier transform basis. Under normal operation, these vectors are not subject to change from iteration to iteration. The MM model computes these space-consuming vectors on the fly for every iteration. However, in the MC model, the vectors are already calculated in an initial phase and saved into special memory locations to access easily whenever needed without recalculation. Thus, the MC model compromises memory space for faster execution time. In the event of node failures, the MC model will be forced to recalculate all values in these vectors. Node failure increases the distance between nodes so new steering vectors need to be established based on new parameters. A new inverse Fourier transform basis is also necessary when the processing frequency bins are changed. By contrast, the MM model encounters significantly fewer disturbances in the event of failures since it would have recalculated all values in any case.

Due to the recalculation in the steering and inverse Fourier transform stages, execution times of these stages are severely increased for the MM model. Conversely, required memory space for these stages is significantly smaller than that required in the MC model. Experiments were conducted to examine the execution time and memory requirements of the two sequential models. The platform used was a SPARCstation-20 workstation with an 85MHz SuperSPARC-II processor and 64MB of memory, and

running Solaris 2.5. Figure 3.5 shows average execution time per iteration for 1000 iterations, and required memory space, for the SA-CBF with 16 nodes, 45 steering angles, 177 output angles, and 4-byte floating-point values. The experimental results represent that execution time of the MC model is five times less than that of the MM model with twice the memory required. The execution time and memory requirement of the FFT stage are the same for both models due to the fact that the same FFT algorithm is used in both. The sub-array summation and cross-correlation/SCOT stages are also unaffected by the model since these stages have no vectors that are the same from iteration to iteration.

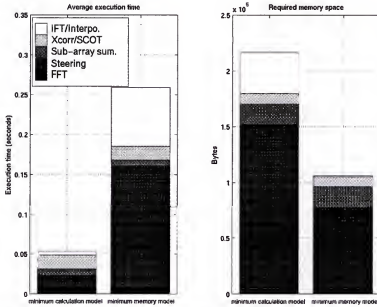


Figure 3.5. Average execution time and memory requirement for MM and MC model

The model selected for the baseline of the performance analysis of parallel algorithms depends on the focus of the study. In the next section, to estimate attainable speedup with each of the parallel programs, execution time is the most important factor to

be measured. Therefore, the MC model is preferred as the sequential baseline. All parallel algorithms are implemented with the same MC model, and it is assumed that each processor has sufficient memory to hold the program and all data.

### 3.3. Parallel Algorithms for the SA-CBF

The best performance of a parallelized task is achieved by minimizing processor stalling and communication overhead between processors. With a homogeneous cluster of processors, dividing tasks evenly among processors serves to maximize performance by reducing these hazards. While task-based parallelism is possible with the SA-CBF algorithm (via assigning steering to one node, sub-array summation to another node, etc.), the workload would not be homogeneous and would result in degraded performance. Figure 3.6a shows this unbalanced workload amongst the various sequential tasks and serves as a justification for not using task-based parallelism.

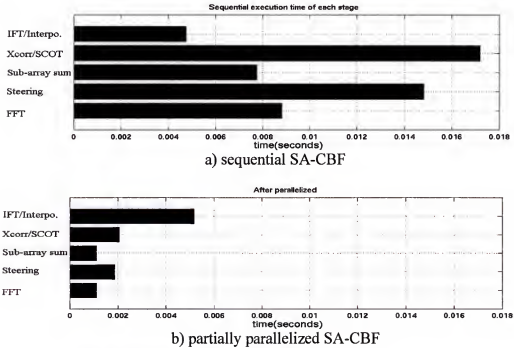


Figure 3.6. Averaged execution time comparison for SA-CBF in an 8-node configuration



The SA-CBF algorithm computes many vector and matrix operations using nested loops. Therefore, we can partition iterations of the outer loop across the processors to create a balanced workload for each processor. When loop partitioning is applied to the parallel SA-CBF algorithm, special attention is required because there are two external loops that are repeated: number of steering angles and number of output angles, as shown in Figure 3.4. These loops run separately within the SA-CBF algorithm but their information is tightly coupled and must be used together to generate the final beamforming output. If we parallelize only the first loop and not the second, which includes the inverse Fourier transform and interpolation stages, severe performance slowdown would result as the number of processors increases. Figure 3.6b shows how loop partitioning of the stages in the first loop results in the second loop becoming a bottleneck. As the number of processors increases, the execution time of each stage decreases linearly except that of the IFT/Interpolation stage since this stage is not parallelized. Since the total execution time of the beamformer in this figure is found by summing the execution times of the stages, it becomes clear that the execution time of the IFT/Interpolation stage will become increasingly dominant as the number of processors increases. Of course, according to Amdahl's law, a small number of sequential operations can significantly limit the speedup achievable by a parallel program [AMD67]. Though parallelization of the IFT/Interpolation stage is nontrivial due to strong dependencies with previous stages, it will significantly increase the efficiency of the overall algorithm.

The two parallel algorithms presented in the rest of this section make use of loop partitioning in two different ways. Furthermore, these algorithms parallelize the second

loop (i.e., the IFT/Interpolation stage) to achieve better efficiency. The next two sections present an overview of the two parallel algorithms, followed by performance results in Section 3.4.

### 3.3.1. Iteration Decomposition

The first decomposition method involves the partitioning of iterations, the solutions of a complete beamform cycle. Iteration decomposition is a technique whereby multiple beamforming iterations, each operating on a different set of array input samples, are overlapped in execution by pipelining. The algorithm follows the tradition of overlapped concurrent execution pipelining, where one operation does not need to be completed before the next operation is started. The beamforming task for a given sample set is associated with a single node in the parallel system. Other nodes work concurrently on other iterations. Pipelining is achieved by allowing nodes to collect new data from the sensors and begin a new iteration before the current iteration is completed. At the beginning of each iteration, all nodes stop processing the beamforming iterations assigned them just long enough to execute the FFT on their own newly collected samples and send the results to the node assigned the new iteration. Once this data has been sent, all nodes resume the processing of their respective iterations. Using this pipelining procedure, there are as many iterations currently being computed as there are processors in the system, each at a different stage of completion. A block diagram illustrating this algorithm in operation on a 3-node array is shown in Figure 3.7.

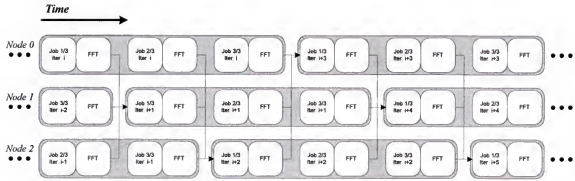


Figure 3.7. Block diagram of the iteration decomposition in a 3-node configuration

Individual beamforming is separated by inter-processor communication stages and each processor takes responsibility for a different beamforming job. A number of difficulties exist for iteration decomposition. First, since every partial job is synchronized at the communication points, an unbalanced processing load can develop across nodes, which may lead to processor stalling. Second, each iteration of the beamforming algorithm must be completed by a node before its pipeline cycle is complete so as to avoid collision between jobs. Therefore, to maximize the performance of the iteration-decomposition algorithm, the beamforming jobs should be evenly segmented by the number of processors. The pseudo-code illustrating the basic algorithm followed by each processor is shown in Figure 3.8.

Each processor calculates an index based on its node number, the current job number, and the number of nodes. This index is used to access arrays which tell the node from which point in its iteration it must continue after executing the FFT and communicating new data and at which point it must again pause in order to begin another new iteration. Specifically, arrays containing the starting steering angle and ending steering angle for a computation block instruct the node on how to partition the first of

the two loops. Arrays containing the starting output angle and ending output angle are used to decompose the second loop. Upon completion of this procedure, a new iteration is started, and each node calculates a new portion of the necessary steering and output angles for its iteration. Managing these arrays requires a nontrivial amount of overhead, but such overhead is a necessary part of the correct operation of the pipelining method.

```

For t=1, Total number of iterations
  For j=1, number of processors //beamforming is divided by # of processors
    index=(j+my_rank)%(number of processors); //my_rank is node number
    Do FFT for their own node data;
    Communicate with other nodes;
    For k=start_steering_angle(index), end_steering_angle(index)
      Steering(k);
      Sub-array summation(k);
      Cross-correlation and SCOT(k);
    End
    For i=start_output_angle(index), end_output_angle(index)
      Inverse Fourier transform and interpolation(i);
    End
  End
End

```

Figure 3.8. Pseudo-code for the iteration decomposition

### 3.3.2. Angle Decomposition

The second parallel algorithm decomposes SA-CBF using a medium-grained approach in which the internals of a complete beamforming iteration are segmented. The angle-decomposition algorithm distributes processing load by decomposing the domain,

the steering angles. Each node calculates the SA-CBF results for a certain number of desired steering directions from the same sample set. Before doing so, all participating nodes must have a copy of the data from all other nodes. After completing this all-to-all communication, each node computes different beamforming angles for the same data. This algorithm introduces considerably more communication than the iteration-decomposition algorithm, and the interconnection scheme between processors will have a more significant effect on performance. The communication requirements are further studied in George et al. [GEO00]. A block diagram illustrating this algorithm in operation on a 3-node array is shown in Figure 3.9.

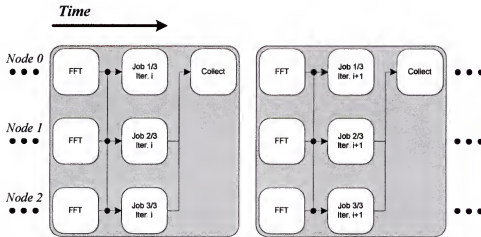


Figure 3.9. Block diagram of the angle decomposition in a 3-node configuration

For the purposes of decomposing the two loops, four variables that indicate beginning and ending steering and output angles are calculated in an initial phase. These four variables serve much the same purpose as the arrays in the iteration decomposition, though for angle decomposition these values are a function only of the node number. The steering direction and the output angle on which a node is to begin computing are

determined by that node's relative location from a virtual front-end node. The number of steering directions a node is to compute is based on dividing the total number of desired steering directions by the number of nodes. The number of output angles per node is also derived from the steering direction information. Figure 3.10 shows the pseudo-code for this approach. After a node is finished computing the results for its steering directions, it must communicate them to a specially designated node for final collection. This collection node can be fixed or, to provide fault tolerance, free-floating perhaps via round-robin scheduling. Although this algorithm involves a more complex communication mechanism best served with a broadcasting network, it does not require the additional overhead necessary to manage pipelining as does the iteration-decomposition method.

```

Calculate angle information based on the node number
For t=1, Total iteration number
    Do FFT for their own node data;
    Communicate with other nodes;
    For k=my_start_steering_angle, my_end_steering_angle
        Steering(k);
        Sub-array summation(k);
        Cross-correlation and SCOT(k);
    End
    For i=my_start_output_angle, my_end_output_angle
        Inverse Fourier transform and interpolation(i);
    End
    Collect result from other nodes;
End

```

Figure 3.10. Pseudo-code for the angle decomposition

### 3.4. Performance Analysis of Parallel SA-CBF Algorithms

In order to understand the strengths and weaknesses of the two parallel algorithms, their performance characteristics were measured on a physical multicomputer testbed. The results of these experiments are presented in this section. The testbed used consists of a cluster of SPARCstation-20 workstations connected by a 155-Mbps (OC-3c) asynchronous transfer mode (ATM) network.

The algorithms were implemented via message-passing parallel programs written in C-MPI (message-passing interface) [MPI94]. In the program code, we call a time-check function at the beginning of a stage and save the return value. After each stage we call the function again, and subtract the earlier return value from the new return value. The difference is the execution time of the stage. In order to obtain reasonable and reliable results, all parallel and sequential experiments were performed 500 times and execution times were averaged.

#### 3.4.1. Sequential Execution Time

The first experiment involves the execution of the sequential SA-CBF algorithm on a single workstation, where the number of sensors is varied to study the effects of problem size. Figure 3.11 shows average execution time per iteration as a function of array size for the sequential SA-CBF algorithm with 500 iterations on the SPARC20/ATM cluster. Execution times of the FFT, steering, and sub-array summation stages increase linearly with an increase in sensors. The number of sensors determines the number of data stream vectors in Figure 3.2a; therefore, the processing load of these stages is greater with increased numbers of sensors. Conversely, the number of data stream vectors remains constant after the sub-array summation stage. No matter how many data stream vectors enter the sub-array summation stage, the number of output data

stream vectors is always two. Furthermore, after interpolation, only one data stream vector is left. Thus, the execution times of Xcorr/SCOT and IFT/Interpolation stages remain fixed as the number of sensors is increased.

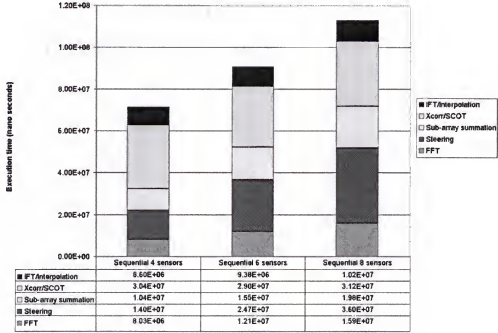


Figure 3.11. Average sequential execution time per iteration

### 3.4.2. Parallel Execution Time

For each of the two parallel algorithms, Figure 3.12 shows average execution time per iteration as a function of array size for the parallel SA-CBF algorithms with 500 iterations on the SPARC20/ATM cluster. For iteration decomposition, the execution time shown represents the effective execution time and not the result latency. As was previously shown for a 4-node configuration, the results for a given beamforming cycle are output after four pipeline stages. In fact, as is typical of all pipelines due to the overhead incurred, this result latency is longer than the total execution time of the sequential algorithm. Instead, the figure plots the effective execution time, which



represents the amount of time between outputs from successive iterations once the pipeline has filled. In the angle-decomposition case, we can measure the execution time directly because there is no overlap of beamforming jobs.

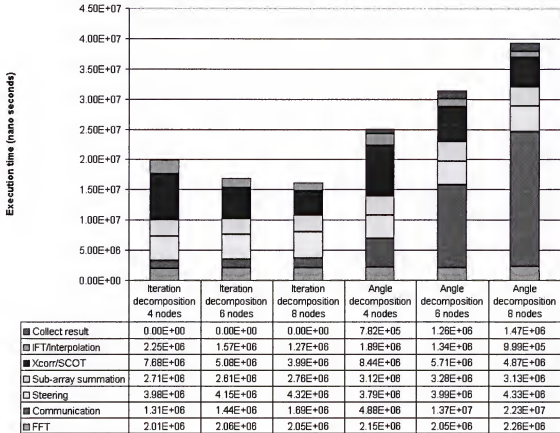


Figure 3.12. Average parallel execution time per iteration

As seen in Figure 3.12, the execution times of the FFT, steering, and sub-array summation stages do not change significantly as the number of nodes increases. As mentioned earlier, the distributed sonar architecture uses smart nodes at each input sensor; therefore, the number of data stream vectors is identical to the number of processors. The additional workload caused by increasing the number of nodes is evenly distributed across the processors. Therefore, the number of nodes does not influence the

execution time of these stages. However, the execution times for the Xcorr/SCOT and IFT/Interpolation stages decrease as the number of nodes and processors increase. In these stages, each processor does less work as the number of nodes increases because the workload from a fixed number of data stream vectors is divided among the processors. In spite of a growing problem size, the overall computation times (i.e. execution time minus communication time) of both decomposition algorithms decline as the number of nodes increases.

In this experiment, communication time is defined as the time spent in communication function calls such as `MPI_Send` and `MPI_Recv`. The communication pattern can be shown with the graphical profiling program Upshot [GRO96]. A sample Upshot output is shown in Figure 3.13, which displays a portion of the profiling log collected from both parallel algorithms running on four nodes in the testbed. In this figure, communication blocks are represented by rectangles defined in the legend, and computation blocks are represented by the horizontal lines between successive communication blocks. Iteration decomposition uses a fairly simple communication pattern but angle decomposition communicates an all-to-all message at the initial phase of each beamforming job. Communication time of iteration decomposition and collection time of angle decomposition have little contribution to total execution time. However, with angle decomposition, the size of the first data communication of each iteration increases rapidly with the number of nodes. With this increase in communication comes an increase in the MPI overhead, an increase in network contention, and poorer performance, which eventually comes to dominate the total execution time. Clearly, the relatively small amount of communication in iteration decomposition is an advantage for

that algorithm. Figure 3.12 indicates that total execution time of iteration decomposition decreases and total execution time of angle decomposition increases with an increasing number of nodes.

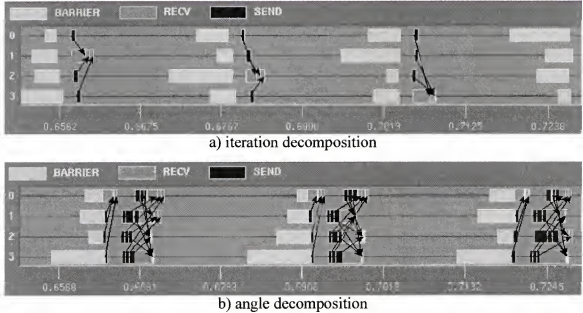
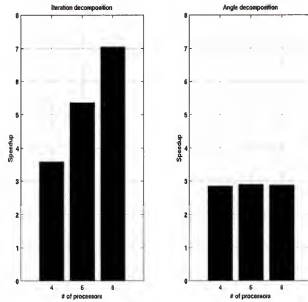


Figure 3.13. Upshot profiles for both of the parallel SA-CBF algorithms with 4 nodes

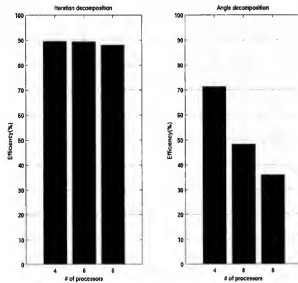
### 3.4.3. Scaled Speedup and Parallel Efficiency

Figure 3.14a shows the scaled speedup of the two decomposition methods over the testbed cluster of SPARCstation-20 workstations. The baseline for comparison is the sequential SA-CBF algorithm running on one workstation of the same testbed. Since the algorithms incur additional workload as sensors are added, the plots show scaled speedup. Figure 3.14b displays scaled efficiency which is defined as scaled speedup divided by the number of processors used. Together these figures show the pronounced effect of the communication overhead in the angle-decomposition method, whereas iteration decomposition exhibits near-linear scaling. Improvement of the performance of angle decomposition may be achieved by employing more complex network

architectures, such as broadcast-efficient networks, and by implementing more robust communication pipelining. Unfortunately, these methods may be impractical to implement on a distributed sonar array topology with limited communication capabilities.



a) scaled speedup



b) scaled parallel efficiency

Figure 3.14. Performance of parallel SA-CBF algorithms

### 3.4.4. Data Memory Capacity

In the previous section, we chose the MC model as the baseline for this investigation. In this model, the majority of the memory requirement arises from the steering stage, as shown in Figure 3.5. Iteration decomposition requires the full amount of steering-vector and steered-signal storage because each processor implements a whole beamforming task for an incoming data set. By contrast, angle decomposition needs only part of the memory space for steering since individual processors generate only part of the beamforming result for a given data set. Figure 3.15 shows memory requirement of the steering stage as a function of the number of processors for both parallel algorithms.

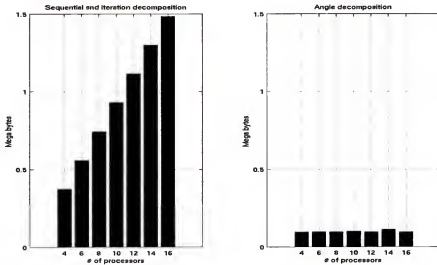


Figure 3.15. Memory requirement of the steering stage

The memory requirements for the sequential algorithm are comparable to those of iteration decomposition. For both the sequential algorithm and iteration decomposition, the demands for memory space for the steering stage grow linearly when the number of nodes is increased, as shown in Figure 3.15. However, little change is observed for angle decomposition. These results illustrate the significant trade-off of execution time versus

memory requirements. Despite the fact that iteration decomposition shows considerably better performance than angle decomposition, angle decomposition may be preferred when memory requirements are stringent.

### 3.5. Summary

The iteration-decomposition algorithm distributes its job in time space with overlap between processors, and the angle-decomposition algorithm parallelizes its job in processor space. The iteration-decomposition algorithm for SA-CBF shows more than 80% scaled efficiency. However, as the number of nodes is increased, the performance of the angle-decomposition method begins to worsen due to inefficiency in the communication stages. In fact, the communication time is the most significant difference between the two parallel algorithms, whereas little difference is observed in computation times. Of the two algorithms, iteration decomposition would be the better choice in architectures with enough memory for each processor to accommodate the large memory requirements. Furthermore, due to the limited amount of communication in iteration decomposition, it would also be well suited for architectures with a low-performance network. By contrast, for a system with a higher-performance network but restricted on memory capacity, angle decomposition may be the better choice. Because angle decomposition uses an all-to-all communication in each iteration, an architecture using an efficient broadcast network would also improve its performance relative to iteration decomposition.

## CHAPTER 4

### PARALLEL ALGORITHMS FOR SUBSPACE PROJECTION BEAMFORMING

The CBF algorithm is quite vulnerable to the noise of incoming data due to the substantial power of the sidelobes in the beampattern known as “spectral leakage.” Frequently, the CBF output in the look direction can be contaminated by unwanted signals that come from directions other than the look direction. Since the level of sidelobes and the width of the mainlobe are proportional to the number of nodes, the CBF algorithm generates a very wide and smooth output with a limited number of nodes and, as a result, a large number of nodes is required for the CBF to resolve the DOA with precise accuracy. The ABF techniques improve the performance of the beamformer by maximizing the signal and minimizing noise power of steered direction output. With certain constraints, the ABF algorithm optimizes the cost function for better beamforming output with narrower and sharper peaks. These improvements generally require extra manipulations to compute the data-dependent weights. Many ABF algorithms have been proposed in the literature [STE89,CAS95,ZHA97,KRO89], but this paper focuses on subspace beamforming algorithms such as multiple signal classification (MUSIC), as presented by Schmidt [SCH81].

Subspace beamforming algorithms exploit properties of eigenstructures to provide a solution to an underlying estimation problem for a given observed process. The significant attention given to the subspace approach in the literature is primarily due to the introduction of the MUSIC algorithm. The performance improvement of the MUSIC

algorithm was so significant that it became an alternative to most existing methods [KRI96]. Stoica and Nehorai [STO89] mathematically analyzed the performance and derived the Cramer-Rao bound (CRB) of the MUSIC method.

The MUSIC beamformer requires eigendecomposition of the CSM to separate the signal and noise subspaces. One method for eigendecomposition is singular value decomposition (SVD); however, the computation of the SVD is intensive to perform. To implement the MUSIC beamforming algorithm in real-time, considerable processing power is necessary to cope with these demands. A beamformer based on a single front-end processor may prove insufficient as these computational demands increase; thus, several approaches to parallelization have been proposed. Hsiao and Delosme [HSI96] proposed parallel SVD algorithms using a systolic array architecture known as the “coordinate rotation digital computer” (CORDIC). Robertson and Phillips [ROB91] built a MUSIC beamforming algorithm for a pipelined systolic array. Both approaches use iterative methods to compute eigenvalues and eigenvectors of the input data. In a parallel implementation, iterative methods are not desirable since the convergence rate of the algorithm depends on the data characteristics and the distribution of the singular values. This unpredictable workload may develop into an unbalanced workload between processors and lead to performance degradation of the parallel algorithms. For that reason, both the algorithms use a strategy of stopping after a predetermined number of iterations based on some experimentation. In a scalable system, however, where the number of sensors is changed by fault or task requirement, the altered problem size may affect the predetermined number of iterations. Instead of using the SVD algorithm,



Smith and Proudler [SMI96] proposed the subspace projection beamforming (SPB) algorithm using row-zeroing QR decomposition, which has fixed computational load.

The work presented in this paper extends the SPB algorithm for parallel in-situ processing on a linear processor array connected by a network. Parallel experiments were performed on a Linux PC cluster, which follows the Beowulf style [STE95] of parallel computing clusters. Performance analysis of the computational stages of a sequential version of SPB is shown in order to examine the sequential bottlenecks inherent in the system. In addition, novel parallel algorithms for SPB are designed for use with distributed processing arrays, and their performance is analyzed.

A theoretical background of SPB is presented in Section 4.1 with a focus on digital signal processing. A sequential version of the SPB algorithm is given in Section 4.2. In Section 4.3, two parallel SPB algorithms are presented. In Section 4.4, the performance of the parallel SPB algorithms, in terms of execution time, speedup, efficiency, result latency, and memory requirements, is examined. Finally, a summary of the strengths and weaknesses of the algorithms is presented in Section 4.5.

#### 4.1. SPB Algorithm

The SPB is based on the subspace beamforming method, which uses the orthogonal property between signal and noise subspaces. The SPB decomposes the CSM into subspaces via row-zeroing QR decomposition. Each column of the orthogonal matrix  $Q$  contains DOA information. The columns corresponding to the target locations are signal subspace; otherwise, the columns are noise subspace. The reciprocal values of the steered noise subspace are the output of the SPB algorithm. Thus, peak points in the beamforming output indicate DOA of sources.

Unlike the MUSIC algorithm, the SPB algorithm has a fixed amount of computation because the algorithm estimates rank and subspace by QR decomposition. The conventional SVD algorithm for the MUSIC estimator requires three steps to compute eigenvectors and eigenvalues. The input data matrix, the CSM, is defined as

$$C = E\{x \cdot x^H\} \quad (4.1)$$

where  $x$  is a vector of Fourier components near the desired frequency,  $H$  denotes complex conjugation and transposition, and  $E\{\}$  is the expectation operation. The first step for the SVD algorithm of the CSM, a complex Hermitian matrix, is to reduce the CSM to a complex tridiagonal form by Householder matrices. Next, using a complex diagonal matrix, the complex tridiagonal matrix transforms into a real tridiagonal matrix. Finally, by an implicit QL algorithm [MAR68] or QR algorithm, the tridiagonal matrix converges to a diagonal matrix. These diagonal entries of the final matrix are eigenvalues of the CSM, and the product of all transformation matrices is the eigenvector matrix. In the final stage, the QR or QL algorithm is an iterative method with a stop criterion; therefore, the computational load for the SVD is usually unknown.

Smith and Proudler [SMI96] designed the SPB algorithm and showed that the subspace, estimated by row-zeroing QR decomposition, is almost identical with the subspace from SVD when the signal and noise eigenvalues are well separated. A row-zeroing method for QR decomposition was used because small elements on the leading diagonal of the upper triangular matrix  $R$  produce corruption subspaces. Therefore, the  $R$  matrix becomes an unreliable indicator of further rank-deficiency. The decomposition avoids the problem by zeroing  $R$  rows, which have a small difference between previous

subspaces or a small leading diagonal element. The main focus of the algorithm is to estimate rank and signal subspace by row-zeroing QR decomposition, as shown by:

$$\Sigma_S \Sigma_S^H C \approx Q_S Q_S^H C \quad (4.2)$$

where  $\Sigma_S$  is a signal column subspace from SVD and  $Q_S$  is columns of the  $Q$  matrix, which corresponds to the signal subspace.

In this realization, we assume that the number of signals is known, and the number of nodes is larger than the number of signals. Generally, the number of signals is determined directly by evaluating the eigenvalues of CSM or, in the SPB algorithm, the number can be obtained by assessing the diagonal elements of  $R$  matrix.

For simplicity, QR decomposition is realized by the Householder method in this SPB implementation. The Householder method is easily extended to row-zeroing QR decomposition. Also, the QR decomposition is equivalent to, and provides an efficient implementation of, the Gram-Schmidt orthogonalization (GSO) process, which is used by the original SPB algorithm. In addition, the number of multiplications required to compute  $R$  in the QR decomposition using the Householder method is about half the multiplication count of the Givens QR method [CUL94].

The basic idea and implementation of the Householder method are explained by several references such as Golub and Van Loan [GOL96]. The QR decomposition based on the Householder method entails  $N-1$  reflections, which is the matrix multiplication between the Householder matrix and the input matrix, to annihilate subdiagonal elements of the  $N \times N$  CSM matrix. Each matrix multiplication zeros out subdiagonal elements of the  $k$ th column of the CSM by multiplying the CSM with  $H_k$ , the Householder matrix. Since the last column of the CSM does not have subdiagonal elements, no reflection is

applied. After  $N-1$  reflections, the CSM is the upper triangular matrix  $R$  and multiplication of all reflection matrices,  $H_k$  is the orthogonal matrix  $Q$  as shown in Equation 4.3.

$$\begin{aligned} H_{N-1} \cdots H_2 H_1 C &= R \\ Q^H C &= R \Rightarrow C = QR \end{aligned} \quad (4.3)$$

Multiplying an elementary reflector  $H_k$  is equivalent to reflecting a matrix across the plane perpendicular to the  $w_k$  vector. For a complex number matrix, modification is necessary for the  $w_k$  vector [HAG88], as shown in Equation 4.4.

$$w_k = \frac{1}{\sqrt{2r(r + |c_{k,k}|)}} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ c_{k,k+1} \\ \vdots \\ c_{k,n} \end{bmatrix} \quad \text{where } r = \sqrt{|c_{k,k}|^2 + |c_{k,k+1}|^2 + \cdots + |c_{k,n}|^2} \quad (4.4)$$

In this equation,  $w_k$  is a vector for the Householder matrix  $H_k$ , which annihilates the  $k$ th column of the CSM, and  $c_{i,j}$  represents the element at the  $i$ th row and  $j$ th column of the CSM. The parameter  $\sigma$  is  $c_{k,k} / |c_{k,k}|$  if  $c_{k,k}$  is nonzero and 1 otherwise. Based on the  $w_k$  vector, the Householder matrix is a matrix of the form below.

$$H_k = I - 2w_k w_k^H \quad (4.5)$$

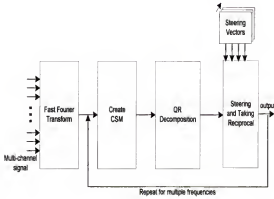
The CSM can now be considered as the QR decomposition rewritten in the form, Equation 4.6.

$$C = QR = [q_1 \ q_2 \ q_3 \ \cdots \ q_n] \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & \cdots & r_{2n} \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & r_{nn} \end{bmatrix}, \quad Q \Rightarrow [Q_S \ Q_N] \quad (4.6)$$

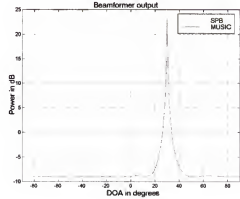
The orthogonal subspace can be divided into signal and noise subspace by comparing diagonal entries of the  $R$  matrix. Provided the matrix  $Q_N$  consists of a different subset of columns of  $Q$ , which exclude signal subspace columns, then the SPB output can be written as

$$P_{SPB}(\theta) = \frac{1}{s^H(\theta) Q_N Q_N^H s(\theta)} \quad (4.7)$$

where  $s(\theta)$  is the steering vector corresponding to look direction  $\theta$ . The noise subspace spanned by the specific columns of the  $Q$  matrix is orthogonal to the signal direction; hence, the multiplication with the steering vector corresponding to the signal direction makes the denominator terms decrease. Consequently, the output of the SPB algorithm produces prominent values near source locations.



a) block diagram



b) sample output with source at  $\theta=30^\circ$

Figure 4.1. Sequential SPB algorithm

To illustrate the above processes, Figure 4.1a shows the block diagram of the SPB algorithm, and Figure 4.1b is a sample output of SPB and MUSIC algorithms with an 8-node configuration. As the original SPB paper [SMI96] showed, the result illustrates that

both beamforming algorithms generate approximately identical outputs at high signal to noise ratio (SNR) scenarios. In the case shown, the SNR is 25dB with white gaussian noise.

The number of emitters we can detect is restricted by the number of input nodes for subspace beamforming algorithms because of the limitation of signal subspace dimensionality. It is desirable to increase the number of input nodes and the processing frequencies in a beamforming system, which generally increases the performance from a statistical perspective, for instance CRB, statistical efficiency [STO89], and so forth. Even if there are enough nodes to obtain a high-quality beamforming output, the number of frequency bins remains an important factor for a high-performance beamformer. For instance, beamformer post-processing often requires an augmented number of processing frequencies because of target classification (i.e. signature) and detection likelihood.

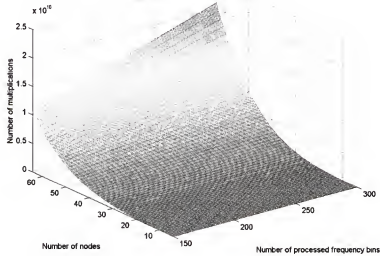


Figure 4.2. The required number of multiplication operations

As both of these parameters increase, the number of multiplication operations required to generate the beamforming output increases rapidly, as shown in Figure 4.2. Due to the QR decomposition stage, the number of multiplications is more sensitive to the number of nodes. This result is expected, since the computational complexity of QR decomposition by the Householder method is  $O(N^3)$ . A more detailed analysis of sequential SPB complexity will be presented in the next section. According to Figure 4.2, significant processing power is essential to generate high-performance beamforming output with acceptable latency and throughput. In cases where current technology cannot provide sufficient real-time performance in a single processor, a trade-off is required between response time and performance. The scalable performance provided by parallel processing will help to overcome limits imposed by a single front-end processor.

#### 4.2. Analysis of the SPB

The SPB algorithm consists of four separate stages: FFT, CSM, QR decomposition, and steering. Each stage has a distinctive function and computational complexity. The FFT stage transforms data from time domain to frequency domain by multiplying with a complex number basis. This domain transformation allows implementation of time shifting in the steering stage by complex number multiplication. The computation complexity of each node's FFT is  $O(N \log_2 N)$  in terms of data length; however, in terms of the number of sensor nodes, the complexity is  $O(N)$ . The creation of the CSM involves an infinite length of computation, which is not feasible in real situations. The CSM is estimated from the input streaming data and updated at each iteration with an exponential average (where  $\alpha$  is the forgetting factor) shown below:

$$\hat{C}(n+1) = \alpha \hat{C}(n) + (1 - \alpha) \{x(n+1) \cdot x(n+1)^H\} \quad (4.8)$$

In Equation 4.1, the expectation is computed by performing the exponential average, which is an estimate of the CSM, assuming that it is stationary. The computational complexity of the CSM stage is  $O(N^2)$  due to the vector-vector multiplication, where  $N$  is the number of nodes. The next stage is to extract orthogonal signal and noise subspaces from the CSM by the QR decomposition. The QR decomposition by the Householder method entails  $N-1$  time reflections to annihilate subdiagonal elements of the CSM. Each reflection creates a Householder matrix, which multiplies with the CSM in consecutive fashion. The QR decomposition stage is the most intensive stage in the SPB algorithm with  $O(N^3)$  complexity. The final stage, steering, computes the output for each direction with complex number vectors. Although the number of steering angles is often considerable, the computational complexity of the steering stage is  $O(N^2)$ . Overall, the computational complexity of the SPB algorithm is bounded by the most intensive stage, QR decomposition; therefore, the sequential complexity is  $O(N^3)$ .

In the sequential SPB algorithm, the number of nodes and number of processed frequency bins play important roles in the computational complexity perspective. However, the number of steering angles is considerably less significant, since the number of steering angles affects only computation at the steering stage. The computational pattern of SPB for multiple frequency bins is identical for each frequency except frequency selection in the FFT stage. For the SPB output, therefore, the beamforming process is repeated for the number of frequency bins. The complexity, which depends on the number of frequency bins, is only  $O(N)$  because of this simple replication processing. Figure 4.2 confirms this observation by increasing linearly in terms of the number of



processed frequency bins. However, the exponential growth of multiplication represents high order complexity for the number of nodes. The CSM stage and steering stage have the same complexity, but the scalar factors for these complexities are different. Figure 4.3 shows that number of multiplications for each stage with 192 processed frequency bins and 2048 FFT length. However, the numbers from the CSM stage and steering stage increase in a quadratic fashion.

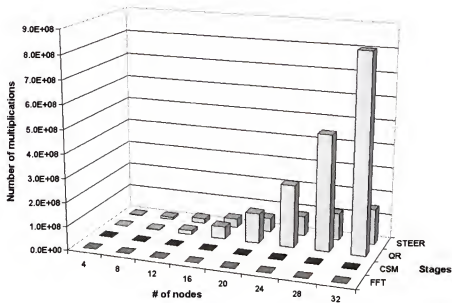


Figure 4.3. Number of multiplications for each stage

In this experiments, a MC model [GEO99], which eliminates redundant computation by using more memory, is selected as the sequential baseline. The sequential SPB algorithm is optimized for computational speed by precalculating the steering vectors and storing in memory. The parallel algorithms are also implemented with the same MC model, and it is assumed that each processor has sufficient memory to store the program and data.

### 4.3. Parallel Algorithms for the SPB

The two parallel algorithms presented in this section make use of decomposition in two different domains, iteration and frequency. The next two sections present an overview of the two parallel algorithms, followed by performance results in Section 4.4.

#### 4.3.1. Iteration Decomposition

The first decomposition method for the SPB algorithm involves the distribution of iterations, that is, the solutions of a complete beamforming cycle. Iteration decomposition is a technique whereby multiple beamforming iterations, each operating on a different set of array input samples, are overlapped in execution by pipelining as explained at the Section 3.3. A block diagram illustrating this algorithm in operation on a three-node array is shown in Figure 4.4. In the figure, solid arrows indicate interprocessor, all-to-one communication, and shaded boxes represent independent complete beamforming cycles. The index in the upper portion of some boxes shows the workload index (e.g. where “1/2” signifies the first of two workload elements).

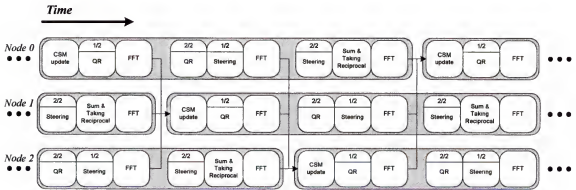


Figure 4.4. Block diagram of the iteration decomposition in a 3-node configuration

For even distribution of workload, iteration decomposition divides the outermost loop of the sequential algorithm across the set of processors. Each divided loop includes all computational stages; therefore, the intermediate result from each computational stage is forwarded to generate a fraction of the final result in each pipelined stage. If one of the computational stages cannot produce the intermediate result, extra pipelining is required to overlap the execution [GEO01]. Unlike SVD and matrix inversion, the intermediate result of QR decomposition can be obtained as the computation is progressing. In the middle of the QR decomposition stage, one column of the  $Q$  matrix and one column of the  $R$  matrix can be obtained by using the matrix reflection. Then, the column of the  $Q$  matrix from the QR decomposition stage is evaluated and steered in the following steering stage. Consequently, the workload distribution in the iteration decomposition is based on the columns of the  $Q$  matrix.

In each pipelined stage, one reflection of the QR decomposition is executed except the final pipelined stage. The  $(N-1)$ th reflection creates the last two columns of the  $Q$  and  $R$  matrices, and there is no necessity for another reflection. So, the DOP for the QR decomposition is not the same as the number of nodes. However, in the final pipelined stage of each iteration, the results of each steered column are added and inverted for the final SPB output.

Each column of the  $Q$  matrix is evaluated to be divided into subspaces by comparing the leading diagonal entry of the  $R$  matrix. If a column of the  $Q$  matrix belongs to the noise subspace, then the column is steered at the next stage; otherwise, the column is discarded. The number of steering stages in iteration decomposition is determined by the number of signals. The CSM stage is performed only at the first

pipelined stage; however, the computational load offered by this stage is not significant, compared with other stages, as shown in Figure 4.3. Therefore, this slight imbalance is not expected to trigger serious computational bottlenecks in iteration decomposition.

#### 4.3.2. Frequency Decomposition

The second decomposable space of the SPB algorithm is the frequency domain. The SPB algorithm generates multiple frequency results by independent computation between frequency bins. The only differences in computation occur in the frequency selection and steering vectors after the FFT stage. The other stages are identical from frequency to frequency with different frequency samples. Therefore, the frequency decomposition algorithm distributes the processing load by decomposing the frequency bins. Each node calculates the SPB results for a certain number of desired frequency bins from the same sample set. Before the start of processing, all participating nodes must have a copy of the proper frequency data from all other nodes. After completing this all-to-all communication, each node computes beamforming for a different frequency from the same data set. A block diagram illustrating this algorithm in operation on a 3-node array is shown in Figure 4.5. In the figure, solid arrows indicate interprocessor, all-to-all communication and shaded boxes represent independent complete beamforming cycles.

The communication requirement of the frequency decomposition is considerably higher than that of the iteration decomposition algorithm due to the all-to-all communication to distribute the data and all-to-one communication to collect results. The second communication is necessary to gather results because each node has partial results of wideband output following the frequency decomposition beamforming computation. The complexity and the number of instances of communication per iteration increase the total execution time of the parallel algorithm.

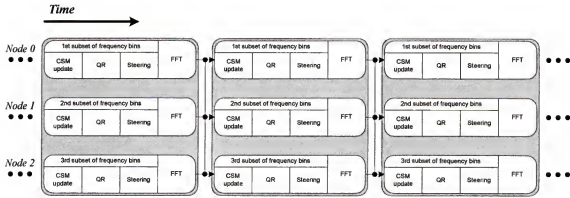


Figure 4.5. Block diagram of the frequency decomposition in a 3-node configuration

To lower the impact of communication in the frequency decomposition method, data packing is used where nodes combine the result data of the previous iteration and the new data for the current iteration as one packet. Data packing eliminates the need to perform an all-to-one communication at the collecting stage between each pipeline stage. The beamforming results of all frequency bins are available at every node after all-to-all communication. For example, the result of the first beamforming iteration is sent by the next instance of communication in Figure 4.5. The use of data packing makes the granularity of the parallel algorithm more coarse. Due to the overhead in setting up communication in a distributed-array system, sending small amounts of data results in a high overhead to payload ratio; hence, it is desirable to combine multiple data packets together to amortize the overhead and reduce the effective execution time. This parallel algorithm involves a more complex communication mechanism best served with a broadcasting network. However, it does not require the additional overhead necessary to manage pipelining, as does the iteration decomposition method.

#### 4.4. Performance Analysis of Parallel SPB Algorithms

In this section, the two parallel algorithms presented are evaluated on a distributed system testbed to analyze the parallel performance. The target testbed of this experiment is a cluster of 32 Linux PCs where each node contains a 400MHz Intel Celeron processor and 64MB of memory. The communication channel between the computers is provided by switched Fast Ethernet.

The algorithms were implemented via message-passing parallel programs written in C with the message-passing interface (MPI) [MPI94]. In the program code, a time-check function is invoked at the beginning of a stage that stores time-stamp information with clock-cycle resolution. After each stage, the function is invoked again and the earlier value is subtracted from the new return value, where the difference is the number of clock cycles required in the execution of the stage. In order to obtain reliable results, all experiments were performed for 900 iterations, and execution times were averaged.

##### 4.4.1. Sequential Execution Time

The first experiment involves the execution of the sequential SPB algorithm on a single computer, where the number of sensors is varied to study the effects of problem size. Figure 4.6 shows average execution time per iteration as a function of array size for the sequential SPB algorithm with 900 iterations on the testbed. The basic beamforming parameters of this experiment are 181 steering angles, 2048 FFT length and 192 frequency bins. Execution times of the FFT, CSM, QR decomposition, and steering stages are observed to increase correspondingly with an increase in the number of sensor nodes. As expected, the execution time of the QR decomposition stage increases rapidly because the complexity of the stage is the most intensive in the SPB algorithm. However, the CSM stage shows less computation time than the FFT stage even though

the CSM stage has a higher computational complexity. The reason for this phenomenon is that, for this problem size, the scalar factor of the FFT complexity is significantly bigger than that of the CSM with a fixed FFT length, 2048. Eventually, the computation time of the CSM stage will exceed the FFT time if the problem size is increased further.

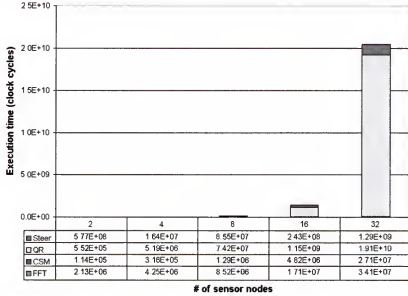


Figure 4.6. Average sequential execution time per iteration

#### 4.4.2. Parallel Execution Time

Figure 4.7 illustrates average parallel execution time as a function of array size for the parallel SPB algorithms with 900 iterations on the Linux PC cluster. The execution time measured from both decomposition methods is the effective execution time, which represents the amount of time between outputs from successive iterations once the pipeline has filled.

The execution time of the FFT stage does not change significantly as the number of nodes increases. As mentioned earlier, the parallel computing model of this

implementation uses intelligent nodes at each input sensor; therefore, the number of data stream vectors is identical to the number of processors. This linear processor model decreases the degree of the complexity polynomial by one in terms of number of nodes. For example, the FFT complexity in this implementation will be  $O(1)$  from  $O(N)$ ; hence, the additional workload caused by increasing the number of nodes is evenly distributed across the processors in this case. As a result, the number of nodes does not influence the execution time of this stage. The execution times for the CSM, QR, and steering stages increase in the fashion of  $O(N)$ ,  $O(N^2)$  and  $O(N)$ , respectively.

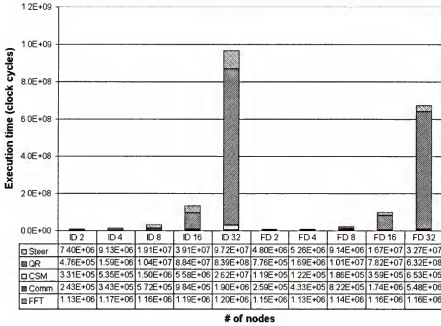


Figure 4.7. Average parallel execution time

Total computation time, which excludes communication time, shows that the iteration decomposition method requires more computation than frequency decomposition. To manage the pipelined execution, the iteration decomposition involves

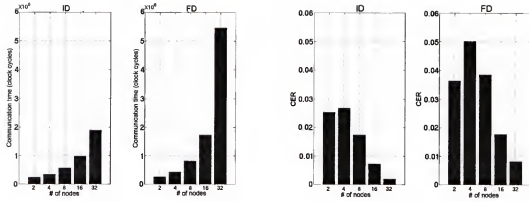


overhead, which allows the sequential workload to be distributed evenly over pipeline stages. In addition to the pipeline overhead, the slight imbalance of workload, as explained in Section 4.3, also increases the effective execution time of iteration decomposition.

#### 4.4.3. Computation vs. Communication

The communication time in this parallel experiment is defined as the time spent in communication function calls of MPI. The iteration decomposition uses a simple all-to-one communication but frequency decomposition communicates an all-to-all message at the initial phase of each beamforming iteration. All nodes require approximately the same amount of time for all-to-all communication, but in all-to-one communication, only one node requires greater communication time for receiving data from other nodes. Since every partial job is synchronized at the communication points, receiving time is considered as communication time for iteration decomposition.

Compared to frequency decomposition, the communication time of iteration decomposition has a smaller contribution to the total execution time. Due to the increased communication complexity in frequency decomposition, the time of the data communication increases rapidly with the number of nodes as evidenced in Figure 4.8a. With this increase in communication comes an increase in the MPI overhead, an increase in network contention, and poorer performance, which eventually deteriorates the parallel performance as the problem and system size increase. Obviously, the relatively small amount of communication in iteration decomposition is an advantage.



a) communication time per iteration

b) communication to execution ratio

Figure 4.8. Communication of the parallel SPB algorithms

The parallelization of the sequential algorithm inevitably requires interprocessor communication, which cannot be divided into any domain. By contrast, it is possible to distribute the computation portion of the algorithm into processors or time space. According to Amdahl's law, a small number of sequential operations can significantly limit the speedup achievable by a parallel program [AMD67]. Thus, the bottleneck caused by the communication limits the speedup if the communication to parallel execution time ratio is significant. To analyze the parallel algorithm from this viewpoint, Figure 4.8b plots the communication to execution time ratio (CER). If the computation portion of the sequential algorithm is effectively partitioned, a smaller CER is desirable because less CER indicates the less communication time for the parallel algorithm. With a small number of nodes, the increased CER may cause performance degradation of both the parallel algorithms. Overall, the CER of both decompositions decreases as the number of nodes increases because of computational complexity; hence, we expect the parallel algorithms for SPB to be scalable in this viewpoint.

#### 4.4.4. Scaled Speedup and Parallel Efficiency

Figure 4.9a shows the scaled speedup of the two decomposition methods on the testbed cluster of Linux PCs. The baseline for comparison is the sequential SPB algorithm running on one PC of the same testbed. Since the algorithms incur additional workload as sensors are added, the plots show scaled speedup. Figure 4.9b displays scaled efficiency, which is defined as scaled speedup divided by the number of processors used. The parallel efficiency for both decomposition techniques demonstrates an increasing trend with a different offset value. For iteration decomposition, the parallel efficiency starts from 45% and levels off at 67%. The efficiency of iteration decomposition is bounded by the pipeline overhead and imbalance of the parallel algorithm rather than by interprocessor communication. The communication time of iteration decomposition is not considerable, as seen from the small CER values; therefore, the contribution of the communication time is negligible. However, in this case, the bottleneck caused by overhead and imbalance limits the speedup to no more than 12 for a 16-node configuration and 23 for a 32-node configuration.

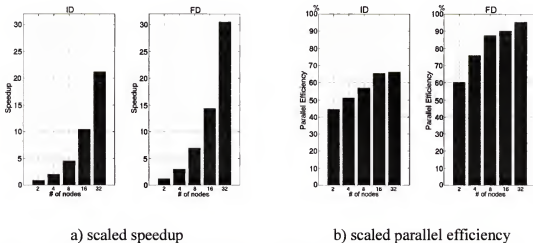


Figure 4.9. Performance of parallel SPB algorithms

For frequency decomposition, the parallel efficiency increases with problem and system size to a greater extent than did iteration decomposition. As mentioned earlier, frequency decomposition efficiently distributes sequential workload with less overhead, but the communication time of the all-to-all scheme compensates for the execution time with a small number of nodes. The advantage of efficient partitioning is not fully realized by the complicated communication up to a certain range of the problem and system size. For the two-node configuration, the parallel efficiency of frequency decomposition is the worst because of the significant communication time. As the number of nodes increases, the computational part becomes more dominant, and the parallel efficiency approaches the ideal. However, it is expected that as the problem and system sizes continue to increase, eventually communication will become a serious performance bottleneck, given the nature of its  $O(N^2)$  communication complexity.

#### 4.4.5. Result Latency

Due to the pipelined property between nodes, the results of iteration decomposition for a given beamforming cycle are obtained after  $N$  stages. As is typical of all pipelines, due to the overhead incurred, the result latency of iteration decomposition is greater than the total execution time of the sequential algorithm. For frequency decomposition, the communication is overlapped across iterations by the data packing technique; therefore, the result latency of this parallel algorithm is the effective execution time plus a small amount of overhead. In this case, after the data are transformed, the first communication takes place to distribute data to all nodes. The generated beamforming results from each node are delivered by the next communication. In addition to the effective execution time, the result latency of frequency decomposition requires one more FFT and communication time. Figure 4.10 shows average result

latency time per beamforming job as a function of array size for the sequential and parallel SPB algorithms with 900 iterations on the Linux PC cluster. As expected, frequency decomposition experiences short latency but iteration decomposition suffers from longer latency than the sequential algorithm.

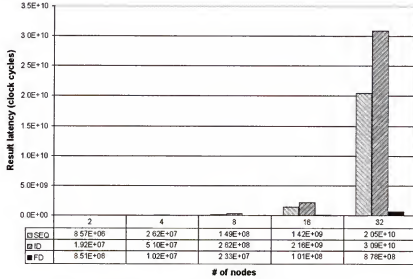


Figure 4.10. Average result latency time per beamforming job

#### 4.4.6. Data Memory Capacity

For faster execution, most of the invariant parameters are stored in memory space. In this configuration, the majority of the memory requirements associated with the sequential and parallel SPB algorithms arises from the steering stage. Iteration decomposition requires the full amount of steering vectors because each processor implements a whole beamforming task for an incoming data set. Therefore, the memory requirements for the sequential algorithm are comparable to those of iteration decomposition. By contrast, frequency decomposition needs only part of the memory space for steering since individual processors generate only part of the beamforming

result for a given data set. For both the sequential algorithm and iteration decomposition, the demands for memory space for the steering stage grow linearly as the number of nodes is increased, as shown in Figure 4.11. However, little change is observed for frequency decomposition.

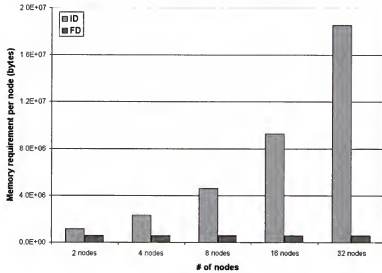


Figure 4.11. Memory requirement of the steering stage as a function of nodes

#### 4.5. Summary

In this paper, two novel parallel algorithms are developed and analyzed for the computationally intensive SPB beamforming algorithm. In the iteration decomposition, the sequential workload is divided into the number of processor stages and these stages are overlapped in execution by pipelining. Each processor in the frequency decomposition calculates the beamforming results for a subset of desired frequency bins from the same data set. For the workload distribution, the iteration decomposition uses the column and row of the sequential matrix computation and the frequency

decomposition is based on the entire matrix computation for the certain set of the frequency bins. Another major difference between the two parallel algorithms is the communication pattern, which is all-to-one for the iteration decomposition and all-to-all for the frequency decomposition. The performance of these parallel algorithms is investigated as aspects of communication and computation such as execution time, scalable speedup, parallel efficiency, result latency, and memory requirements.

Overall, both the parallel algorithms achieve scalable parallel performance with increasing scaled efficiency. On the testbed, it was observed that parallel efficiency increased by almost 22% and 35% from a 2-node to a 32-node system with iteration decomposition and frequency decomposition, respectively. Due to the cubic computational complexity in the sequential algorithm, the partitioned computation in the parallel SPB algorithms occupies most of the parallel execution time as the number of nodes increases. Thus, the overhead caused by parallel implementation is concealed by these computation times and parallel efficiency of both algorithms exhibits better performance for larger system size. Of the two algorithms, frequency decomposition is the better overall choice since it shows better scalable performance with a smaller memory requirement and shorter result latency as ratio of system size. Furthermore, the parallel performance of frequency decomposition may be further improved by using a high-speed network with efficient broadcast capability. By contrast, due to the limited amount of communication, iteration decomposition may be well suited for architectures with a low-performance network, or when problem and system sizes begin to make communication the dominant factor in execution time.

## CHAPTER 5

### PARALLEL ALGORITHMS FOR CONVENTIONAL MATCHED-FIELD PROCESSING

During a long period of time, plane-wave beamforming algorithms have been pervasive as techniques to find the DOA of the target for underwater signal processing. However, under certain situations such as shallow water, plane-wave beamforming increases the likelihood to miss the target or produce misleading target locations because of the mismatch problem of the data model. The simple phased model of plane-wave beamforming is inappropriate to describe the underwater wave propagation. Due to the limited performance of the plane-wave beamforming algorithms, complex-wave propagation based on a more realistic model was introduced to enhance the capability of sonar signal processing. In the new model, the waveguide nature of the ocean and indirect paths between the target and the sensor are considered to describe wave propagation reliably.

These advanced beamforming algorithms from a complex data model exhibit high levels of computational complexity and memory utilization. These hurdles make implementation in real-time sonar array systems a significant challenge. Because of the computational resources necessary to carry out the MFP, it is appropriate to combine these signal processing efforts with modern data telemetry and networking technologies. Taken together, these trends make imperative the development and use of advanced distributed and parallel processing techniques in terms of algorithm, architecture,



network, and system design to reduce the significant processing delay of the MFP algorithms.

The MFP is a generalization of plane-wave beamforming in which the steering vectors are derived from the spatial point source response of the medium. The conventional matched-field processing (CMFP) algorithm was first proposed by Bucker [BUC76] and first implemented by Fizell and Wales [FIZ85]. To determine the location of a source, Bucker's MFP algorithm compares the measured pressure of signals at a receiver to the theoretical transmissions at the source. By contrast, with plane-wave beamforming the image of a point source is reconstructed on the basis of simple time delay; such a model is inadequate in an ocean waveguide. In ocean acoustics, the reflection of sound at the boundaries, represented as a number of discrete arrivals, is measured at the far field of an acoustic source because of waveguide propagation. In the MFP algorithm, the target location is determined by matching the measured acoustic pressure field at the sensor outputs to a predicted pressure field, which is based on an assumed source location. The choice of a suitable acoustic propagation model determines the predicted acoustic field at the sensors. The objective of the propagation model is to examine which analytical waveguide model corresponds to the real waveguide, along with the actual conceptual feasibility of localization of a source in a real waveguide. The experimental possibility of reconstructing and localizing a source in a waveguide, which is described by an analytical model, is confirmed in many papers such as [ZUI93].

For the plane-wave data model, the implied environment is an infinite, isotropic, homogeneous, three-dimensional space, which reduces the wave equation into an

uncomplicated form. The solution for the wave equation is represented by a simple phased model. However, when realistic environments are introduced, there is no easy way to compute how a source at one spot will be heard at another. The sound propagation in a waveguide is an extremely complicated problem and cannot be solved in most cases. The difficulty lies in the fact that the solution must satisfy not only the wave equation, but also the initial conditions and boundary conditions. The wave equation could be solved directly using finite-difference or finite-element methods. In general, these methods are not implemented since such techniques are not computationally feasible on most computers. Depending upon the approach, the solution of the wave equation is classified into many categories, as described by Porter [POR93]. All acoustic models make various simplifying assumptions on the wave equation to extract the predicted pressure field.

By approaching the solution of the wave equation from the waveguide nature and separation of variables, the normal-mode solution is obtained for computational description of the sound field. The normal modes may be considered as standing wave patterns in the vertical direction that move out from the source in the horizontal direction. Normal-mode solutions are not the only acoustic models appropriate to shallow water. The model, however, is well suited to MFP since it needs to be executed only once to generate an entire range and depth surface [VAC98]. Therefore, all the considerations in this research are based on the normal-mode solution to the wave equation for a uniform, rigid, shallow-water channel. The normal-mode solutions provide an estimate for the acoustic sound field given a shallow-water channel where the bottom depth and profile are not a function of range [BUC76]. If the bottom profile changes with range, normal-

mode perturbation theory or some other formulation is required. More precise methods for steering vector computation exist, but usually at the expense of computation. A wealth of information is available on the subject, and the mathematical normal-mode solution is provided by many papers such as [UNI69,URI75,POR93].

The CMFP algorithm requires both significant processing power, as well as memory demands that supercede the capabilities of most current real-time uniprocessor systems. To cope with the computational and storage challenges of the CMFP, efficient distributed parallel algorithms for the CMFP need to be developed. Such parallel algorithms might be expected to provide a feasible solution to real-time, deployable, and cost-effective beamforming by distributing a workload over an array of processors.

To demonstrate the feasibility of the algorithms being developed, a series of parallel experiments are performed on a cluster of personal computers (PCs) and also on an array of digital signal processors (DSPs). Fundamentally, the PC cluster is used in this research to predict the performance and scalability of parallel algorithms. Certain circumstances deem the solution provided by the PC cluster inappropriate because of system complexity, size, and, most importantly, power consumption. Although the PC cluster features flexible configuration with cost-effective scalability, an array of DSPs is the final destination for developed parallel algorithms because of the limitations enumerated. The array employs efficient interprocessor communication using a service known as MPI-SHARC [KOH01], to accelerate complex collective communication. Performance results from the array will be provided in this paper, as well as PC cluster results.

Besides parallel performance parameters described in previous chapters, the interesting benefit of parallel processing is the energy efficiency of the beamforming system. This additional advantage provides critical performance gain in terms of longevity of battery-operated systems, such as portable or rapidly deployable systems. Instead of employing a powerful single processor, the parallel beamforming system consists of multiple processors. The speed of an individual processor in the parallel processing system may be inferior to that of a sequential system, but low-power versions of DSP processors allow the system to consume less energy, as compared to the baseline system. In general, as the speed of a processor is increased, its power usage is also increased proportionally. The parallel configuration also allows the system to consume the power and energy in a scalable fashion by using multiple processors. The total energy consumption of the parallel system is determined by the number of processors involved; therefore, the system can be designed to use a certain amount of energy for the task requirement. In this chapter, the power and energy requirements of parallel beamforming systems will be measured and computed as a new avenue of scalability metric.

Section 5.1 presents the theoretical background of the CMFP algorithm used as a basis for this study, and Section 5.2 analyzes the sequential version of the CMFP algorithm in terms of computation. Section 5.3 is dedicated to the explanation of two novel parallel algorithms for the CMFP. Experimental testbeds used to analyze the parallel algorithms are described in Section 5.4. Section 5.5 then explores and examines the performance of the parallel CMFP algorithms in terms of the extensive set of parameters. Finally, Section 5.6 provides the brief summary of this chapter.

### 5.1. CMFP Algorithm

The MFP localizes a source more accurately than plane-wave methods; in particular, range and depths can be estimated well since they incorporate the full wave physics of the acoustics, including both signal and noise processes, into the array processing. For precise target localization, the propagation model must be accurate; otherwise, the performance of the MFP will suffer from the significant mismatch problems described by Baggeroer et al. [BAG93]. The calculation of the predicted acoustic field is far more difficult than plane-wave beamforming due to the more realistic propagating effect and the need for solving the wave equation given by Equation 5.1.

$$\nabla^2 p(r, z) + \frac{\omega^2}{c^2(r, z)} p(r, z) = \frac{-\delta(r - r_s) \delta(z - z_s)}{r} \quad (5.1)$$

Here,  $\nabla^2$  represents second-order partial derivatives,  $p(r, z)$  is the pressure at range  $r$  and depth  $z$ ,  $c(r, z)$  is the ocean sound speed,  $\omega$  is the angular frequency of the source located at range  $r_s$  and depth  $z_s$ , and  $\delta(x - x_o)$  is the delta function, also known as the unit impulse function. The derivation and physical translation of the wave equations are illustrated in [UNI69]. By solving the wave equation, the acoustic pressure  $p(r, z)$  is obtained as a function of range and depth for the steering vector of the MFP algorithm.

The solution of the wave equation depends on the acoustic models applied. There is no single closed-form solution of the given wave equation due to the initial conditions and the boundary conditions. In the acoustic model, we assume a certain form of solution and wave propagation to meet the conditions. Numerous acoustic models have been proposed to solve the wave equation. Although one model may be able to handle most of the situations encountered, at least some of the cases are usually more efficiently treated by another model. Computational pattern and complexity are quite diverse for each

model as well. The discussion of model performance is beyond the scope of this research; however, a brief explanation of an employed model is noted.

When the signal path can be modeled as a simple waveguide, the normal-mode solution is efficient in terms of model performance and computation. The pressure field at any point in the waveguide can be represented as a sum of vertical standing waves when one source excites the waveguide. The main limitation to the normal-mode solution is that the solution is considered principally in the context of range-invariant conditions. However, the normal-mode solution can be extended in various ways to both range-dependent problems and fully three-dimensional problems [JEN94]. The normal-mode solution begins with the assumption that the solution of the wave equation consists of the product of range- and depth-dependent terms. By separation of variables, the solution of the wave equation is provided as follows:

$$p(r, z) \approx \frac{i}{\sqrt{8\pi r}} e^{\frac{iz}{c}} \sum_{m=1}^{\infty} \Psi_m(z_s) \Psi_m(z) \frac{e^{ik_m r}}{\sqrt{k_m}} \quad (5.2)$$

In the equation,  $z_s$  is the source depth,  $r$  is the range between source and receiver,  $k_m^2$  is a separation constant where  $k_m$  is the horizontal wave number for the  $m$ th mode, and  $\Psi_m(z)$  is the normal-mode depth function corresponding to  $k_m^2$ . The horizontal wave number and normal-mode depth function are calculated from the eigenvalue problem with environmental parameters. The ambient environment data alter the number of parameters and parameter values on the eigenvalue problem. The accuracy of environment information, therefore, is very important. Many existing methods are used for the normal-mode solution. The most widely used method is KRAKEN, which is based on a finite-difference algorithm. The popularity of KRAKEN originates from the matrix that is set up as a tri-diagonal matrix for which very fast eigenvalue and

eigenvector solution techniques are available. The work herein is based on the KRAKEN model developed by Porter and described in [POR91,POR84,JEN94]. Figure 5.1 shows the sample normal-mode intensity plot from the Porter code [POR94] for a wide underwater area with a 1000m-depth source. This figure clearly illustrates the complex-wave propagation in the ocean. In the next section, a further computational method of KRAKEN will be discussed in matrix format.

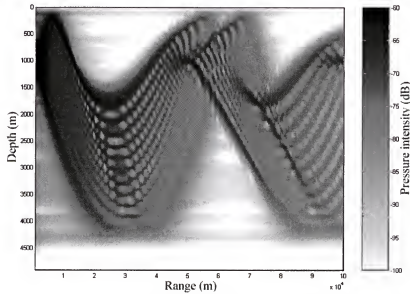


Figure 5.1. Normal-mode propagation with a source at 1000m depth [POR94]

The CMFP is the most fundamental form of the MFP algorithm. Advanced techniques, such as adaptive methods, can be applied in order to improve the MFP output. The CMFP has no computational difference with conventional beamforming [BAR48] after the steering vectors have been computed. A match is formed by testing a set of modeled steering vectors and comparing them to the CSM formed by the signal received at the sensor array. There are generally four important computational stages:

FFT, CSM update, steering, and steering vector generation. The FFT and CSM update stage are identical with those of SPB algorithm. The steering procedure estimates the field distribution versus the sensor input. The stage is both a function of the CSM and the steering vectors. The steering vector generation stage uses normal-mode solution and environmental data to solve the wave equation for the spatial response of a source with a given location. Figure 5.2 illustrates the computational flow of the CMFP.

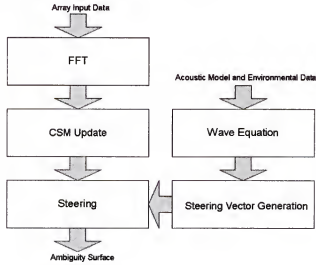


Figure 5.2. CMFP block diagram

To summarize the previous discussion, the equation for the CMFP algorithm is provided below.

$$DF(r, z) = G(r, z)^H \cdot C \cdot G(r, z) \quad (5.3)$$

Here, the  $DF(r, z)$  is the detection factor (ambiguity surface) at range  $r$  and depth  $z$ ,  $R$  is the CSM and  $G(r, z)$  represents the normalized steering vectors for a given range and depth. Note that the normal-mode solution of the wave equation is steering vectors without normalization. The output of the CMFP produces a three-dimensional plot of



detection factors versus range and depth for specific frequency. For broadband processing, the multiple CMFP results are averaged over a wide range of frequency. Independent steering vectors and the CSM are required for each frequency CMFP result. The final resulting surface is commonly referred to as the ambiguity surface and presents the likeliness of detection for a given data set. Peak locations on the plot represent probable source locations. Figure 5.3 shows the CMFP ambiguity surface using the normal-mode solution for a source at a 50m depth and 10Km range. The range/depth plane consisted of 5 to 44Km in range and 0 to 158m in depth. The resolution used was 1Km in range and 2m in depth. The frequency resolution used in the Fourier decomposition was 1.0Hz across a band from 200 to 231Hz. The vertical array contained 32 sensors spaced at 4m with the top sensor at 10m depth. The generated data do not contain any noise and no uncertainty exists about the environment data.

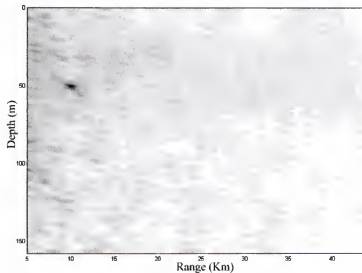


Figure 5.3. Sample output of the CMFP for a 32-node array

The CMFP equation, Equation 5.3, is almost identical to the CBF algorithm, except the steering vectors are now a function of range and depth. The computational pattern of both algorithms is similar. However, the required complexity and size of the steering vectors for the CMFP are much larger than those of the CBF due to the complicated wave equation solution and more resolvable coordinate of steering vectors, such as range and depth. Therefore, the CMFP requires more processing power and memory capacity. The real computational problem with the CMFP is not only requiring more computation to execute, but also that it requires intensive computation and an extensive amount of storage to calculate and preserve steering vectors in the initial phase.

## 5.2. Analysis of the CMFP

As previously mentioned, the CMFP algorithm consists of four distinct computational stages: FFT, CSM update, steering, and steering vector generation. In the initial phase of the algorithm, the FFT is implemented by the radix-2 method based on the butterfly pattern computation. The generic complexity of the FFT is  $O(N \log_2 N)$  in terms of data length. In terms of the number of sensor nodes in the CMFP, however, the complexity is  $O(N)$  because of the simple replication process for each sensor node. The second stage of the CMFP is related to the covariance matrix computation. The CSM is estimated from the input streaming data and updated at each iteration with an exponential average as shown Equation 4.8. The convergence rate for the CSM estimation mainly depends on the data characteristics, such as a time-variant condition and forgetting factor. In this implementation, input data are stationary and time-invariant. The computational complexity of the CSM update stage is  $O(N^2)$  due to the vector-vector multiplication, where  $N$  is the number of sensor nodes.

The next stage, steering, computes the output for each range and depth with complex number vectors, which are steering vectors. Due to the multiplications between steering vectors and CSM as shown in Equation 5.3, the complexity for the steering stage is  $O(N^2)$ . Besides these data-dependent stages, there is the steering vector generation stage, which is invariant to input data. Once the steering vectors are calculated from environmental parameters, the values remain until any ambient changes are observed. The computational procedure of Equation 5.2 for the steering vector generation is explained in [POR97] with matrix format. A brief computational procedure is given in Figure 5.4.

1. Create a tri-diagonal matrix from environmental parameters (speed of sound, sensor configuration, etc.)
2. Solve the eigenvalue problem with the tri-diagonal matrix
3. Sort eigenvalues and keep the modes with the largest real part
4. Normalize the eigenvectors
5. Assemble the eigenvectors into a matrix
6. Calculate other coefficients
7. Sum up the modes for final steering vectors

Figure 5.4. Computational procedure of the KRAKEN normal-mode solution [POR97]

In this computation, several critical parameters are involved, and the performance of the CMFP algorithm relies on the accuracy of the information. Performance variation from uncertain information is illustrated by Gingras [GIN93]. The complexity of this stage can be represented by several parameters. Usually the steering vector generation stage has high order complexity due to the computationally intensive procedures, such as

the eigenvalue problem. However, as with the FFT stage complexity, the steering vector generation stage reveals linear complexity in terms of the number of sensor nodes. The principal philosophy is identical with the FFT stage. Fundamentally, the number of generation processes is the same as the number of sensor nodes; in other words, each sensor node is required to generate the entire range and depth of steering values separately. In spite of the low complexity of the stage, the computational burden presented by the steering vector generation stage is exceptionally significant due to a substantial scalar factor in the complexity. For all problem sizes implemented in this experiment, the most computationally intensive stage is the steering vector generation stage.

Another independent parameter considered is the number of frequency bins. In this investigation, the problem size is scaled by changing the number of sensors, but the number of frequency bins is predetermined. Sometimes it is desirable to increase the number of sensors and the processing frequency bins in a CMFP system, which generally increases the performance from a statistical perspective. Even if there are enough sensor nodes to obtain a high-quality beamforming output, the number of frequency bins remains an important factor for a high-performance beamformer. The postprocessor of the beamformer often requires an augmented number of processing frequency bins due to target classification (i.e. signature) and detection likelihood. The complexity of all computational stages, except the FFT, is linear in terms of the number of frequency bins because of independent computation between frequency bins. After frequency selection at the FFT stage, the beamformer output is first calculated for each individual frequency bin for narrowband processing. The final output for broadband CMFP is obtained by

averaging the outputs obtained for all frequency bins. Hence, as the number of frequency bins is increased, the beamformer has to compute an increased number of narrowband CMFP results. The number of frequency bins is the same as the number of narrowband CMFP computations; therefore, the linear complexity is observed.

The model selected for the baseline of the parallel performance analysis depends on the focus of the study and experimental testbed. For this experiment, we consider two distinct testbeds, which are explained in detail in a later section. One testbed is the PC cluster based on a network of conventional PCs, and the other testbed is an array of DSPs. It is presumed that each PC of the former testbed has sufficient memory to hold entire steering vectors of output points, and DSPs from the latter testbed are memory-bounded types. To estimate attainable speedup with each of the parallel programs, execution time is the most important factor to be measured. The MC model is preferred as the sequential baseline for both testbeds, but the MC model is not realizable in the array because of memory limitations. Therefore, the MC model is applied for the baseline of the PC cluster testbed, and the MM model is employed for the baseline of the array of DSPs.

Although with the MM model the entire set of parameters of should be computed on the fly from environmental data, it is assumed that eigenvalues and eigenvectors are pre-computed and stored in the memory. The eigenvalues and eigenvectors are subject to be invariant unless environmental changes are monitored. From this exception, the tremendous computational burden of the generation stage is reduced to some extent by small memory usage. All parallel algorithms are implemented with the corresponding models for each testbed. The steering stage from the MM model includes the steering

vector generation computation so the workload of the steering stage is significantly increased by this additional computation.

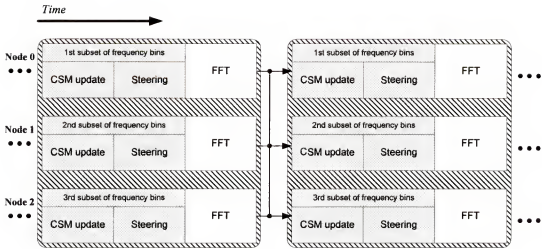
### 5.3. Parallel Algorithms for the CMFP

The two parallel algorithms presented in this section make use of decomposition in two different domains: frequency and output points. The next two sections present an overview of the two parallel algorithms, followed by an overview of the testbeds in Section 5.4 and performance results in Section 5.5.

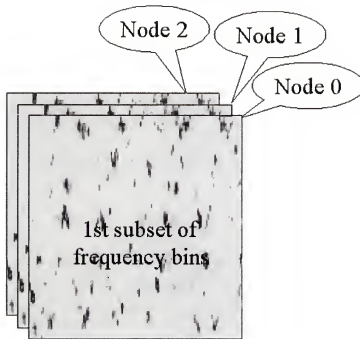
#### 5.3.1. Frequency Decomposition

The first decomposable space of the CMFP algorithm is the frequency domain. The CMFP algorithm generates multiple frequency results by independent computation between frequency bins. The frequency decomposition algorithm distributes the processing load by decomposing the frequency bins. Each node calculates the CMFP results for a certain number of desired frequency bins from the same sample set, as explained in Section 4.3. The workload distribution from frequency basis is illustrated in Figure 5.5b, and a block diagram illustrating this algorithm in operation is shown in Figure 5.5a, both for a 3-node array.

The communication requirement of the frequency decomposition is considerably high due to the all-to-all communication to distribute the data and to collect results. The result collection is necessary because each node has partial results of wideband output following the frequency-decomposed beamforming computation. To lower the impact of communication in the parallel algorithms, data packing is used where nodes combine the result of the previous iteration and the new data for the current iteration as one packet. The use of data packing makes the granularity of the parallel algorithm more coarse.



a) block diagram



b) workload distribution

Figure 5.5. Frequency decomposition

Besides the number of communications, the frequency decomposition has a significantly long message size offered by partial results from the frequency subset. The CMFP produces a three-dimensional plot of detection factors versus the entire range and depth for specific frequency, as shown in Figure 5.5b. Prior to the final result computation, each sensor node needs to send the entire range and depth result since the broadband CMFP result is computed by averaging the results for each physical point over the entire frequency range. The message size of the partial result remains constant for the same output points, but the number of output points could be more than several thousand in general. This parallel algorithm involves a complex communication mechanism best served with an efficient broadcasting network.

### 5.3.2. Section Decomposition

The second parallel algorithm decomposes the CMFP using a coarse-grained approach as well. The communication method is identical with the frequency decomposition. The section decomposition distributes the processing load by decomposing another domain, the output points. Each node calculates the CMFP results for a certain subset of output from the same data. Before doing so, all participating nodes must have a copy of the data from all other nodes. After completing this all-to-all communication, each node performs a given workload based on the output points. A block diagram illustrating this algorithm in operation on a 3-node array is shown in Figure 5.6a, and the conceptual figure for the workload distribution is provided in Figure 5.6b.



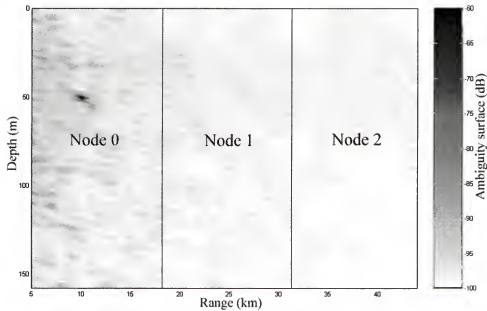
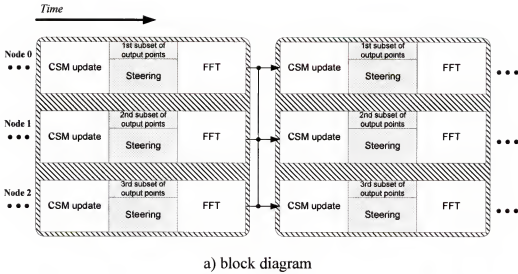


Figure 5.6. Section decomposition

Unlike frequency decomposition, the section decomposition has a dependency between computational stages. Every output result of the CMFP algorithm is an averaged value over the multiple frequency bin results. Therefore, the entire frequency range of the CSM is required for a single output computation. The CSM cannot be explicitly

divided into any domain except frequency bin. We may divide the CSM into an individual subspace by using singular value decomposition at the expense of increased computation. However, in the conventional way, the CSM should not be manipulated in any form. By this dependency, the CSM update stage is excluded from the parallel implementation in section decomposition. The computational load offered by this stage is not significant, compared with other stages. Therefore, this sequential fraction is not expected to trigger serious computational bottlenecks in section decomposition.

The number of output points to compute per node is based on dividing the total number of output points by the number of sensor nodes. After a node is finished computing the results for its output points, the results must be sent by the next instance of communication. The data packing technique is used for section decomposition as well. For frequency decomposition, the message size for all-to-all communication is fixed because the message contains the entire range and depth information from partial results. It is noted that the number of output points, which is the product of the number of range points and the number of depth points, is predetermined and does not change for other problem sizes. By contrast, the message size for section decomposition decreases as the number of sensor nodes increases since the fixed number of output points is divided by increasing the number of sensor nodes. The implementation of the CSM update and the communication message size are major distinctions between the two parallel algorithms, and certain performance variations will be expected.

#### 5.4. Experimental Testbeds

In order to understand the strengths and weaknesses of the two parallel algorithms, their performance characteristics were measured on a physical testbed. Two

types of testbeds will be introduced in the following subsections. Both testbeds have similar configurations with multiple processing units connected by a communication channel. However, the characteristics of the processing units and the type of communication channels differentiate the two testbeds. As for the software, the algorithms were implemented via message-passing parallel programs written in C with the message-passing interface (MPI) [MPI94]. In order to obtain reliable results, each experiment on the PC cluster was performed for 900 iterations, and results were averaged. On the DSP array, a single iteration was measured since this testbed provides deterministic time measurement with no transient overhead from an operating system.

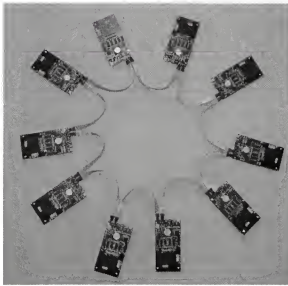
#### 5.4.1. PC Cluster

The first target testbed of this experiment is a Linux-based cluster of 32 PCs where each node contains a 400MHz Intel Celeron processor and 64MB of memory. The communication channel between the computers is provided by switched Fast Ethernet. Version 1.5 of MPI/Pro from MPI Software Technology Inc. provides the messaging layer. The basic configuration of this Linux PC cluster follows the Beowulf style [STE95] of parallel computing clusters, where the emphasis is on the effective use of low-cost, high-performance components.

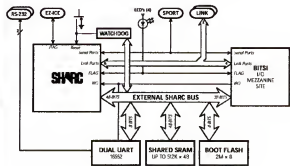
#### 5.4.2. DSP Array

The second target testbed of this experiment consists of 10 Bittware Blacktip-EX DSP development boards [BIT97] connected to one another in a ring topology, as shown in Figure 5.7a. Each board includes a single ADSP-21062 super Harvard architecture (SHARC) processor from Analog Devices, as well as additional hardware for links to other nodes, off-chip memory, and so forth, as illustrated in Figure 5.7b. The SHARC processor, as with many other prevalent DSPs, integrates multiple link ports and their

associated direct memory access (DMA) controllers to provide high-speed, low-overhead communications. The bi-directional ports, each consisting of four data and two handshaking lines, can operate at twice the clock rate of the processor achieving a peak throughput of 40MB/s. Furthermore, the integrated DMA controllers require minimal setup and no processor overhead when communicating with other devices. Finally, the DMA channels assigned to each link port can operate concurrently, allowing both ports to transfer data simultaneously, thereby increasing the aggregate throughput of the system.



a) picture of the DSP array



b) Blacktip-EX development board architecture

Figure 5.7. Digital signal processor array and its node architecture

The development board contains two link ports with external connectors to enable communications with other devices. To eliminate the need for external routing or switching hardware, the two link ports are dedicated to separate send and receive channels. This configuration allows the boards to be arranged in a uni-directional ring topology. Although a ring does not provide the communication scalability of other

topologies, its simple routing and low hardware complexity make it a natural choice for this system. In addition, if developed appropriately, certain message-passing functions can take advantage of a ring topology to produce highly efficient collective communications.

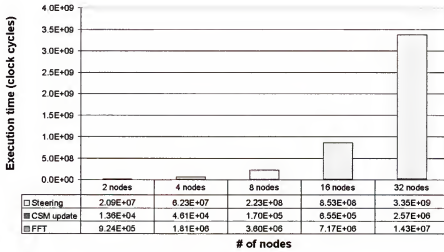
The network service in this DSP array was implemented and optimized for this particular architecture by a previous research effort [KOH01]. To provide the MPI functionality on an array of DSPs, the MPI-SHARC network service was created. Since it was considered impractical to provide all 125 functions defined in the MPI specification on a simple embedded DSP, a reduced version of the specification was examined. The most common functions used by parallel applications were included in the design. Although the MPI-SHARC is a subset of the full specification, the functionality and syntax are identical to the MPI found on common distributed systems, allowing users to easily port applications developed on other platforms to an embedded, distributed DSP system.

### 5.5. Performance Analysis of Parallel CMFP Algorithms

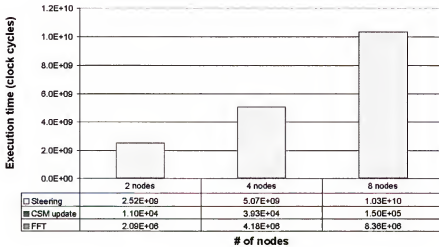
This section explores the performance of the parallel CMFP algorithms on the two distinct testbeds. The measured parameters include execution times, communication times, data memory requirements, and power consumption of sequential and parallel algorithms. Based on these parameters, dependent results can be derived for scaled speedup, parallel efficiency, energy consumption, and scaled energy efficiency. Investigation of these results demonstrates the performance effects of the design issues introduced in the previous section.

### 5.5.1. Sequential Execution Time

The first experiment involves the execution of the sequential CMFP algorithm on a single computer, where the number of sensors is varied to study the effects of problem size. The results from the two testbeds are shown in Figure 5.8. Due to the processing unit limitation in the DSP array, the execution time results are measured up to 8 sensor nodes on that testbed.



a) single PC cluster processor



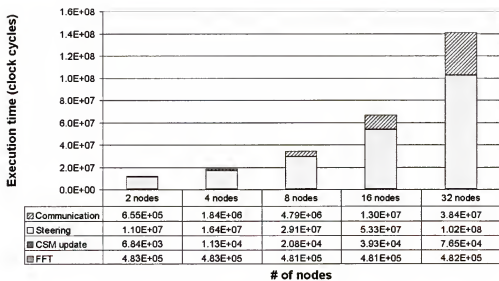
b) single DSP array processor

Figure 5.8. Sequential execution times vs. number of sensor nodes

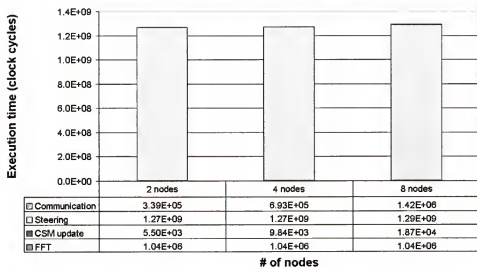
As previously noted, the sequential CMFP for the PC cluster uses the MC model and the sequential algorithm for the DSP array is built on the MM model. The steering vector generation is included within the steering stage in the MM model and the steering stage in the MC model excludes the steering vector generation. In the MC model, the CMFP algorithm computes the output with pre-calculated and stored steering vectors. Thus, the execution times of the sequential CMFP on the DSP array are considerably higher than their counterparts. On both testbeds, execution times of the FFT, CSM update, and steering stages are observed to increase correspondingly with an increase in the number of sensor nodes. The execution time of the steering stage increases rapidly for the PC cluster due to the complexity involved in this stage. Linear growth of steering time on the DSP array is observed due to the significant steering vector generation time. The quadratic complexity of steering is concealed by the enormous linear complexity for the given problem sizes. For the PC cluster, the CSM update stage shows less computation time than the steering stage even though both stages have the same computational complexity. The reason is that the scalar factor of the steering complexity is significantly bigger than that of the CSM update stage due to the substantial number of output points, 3200 in this case (80 vertical points times 40 horizontal points). But the execution time increases of both stages on the PC cluster show quadratic scaling for the increasing problem size.

#### 5.5.2. Parallel Execution Time

Figure 5.9 and Figure 5.10 illustrate the averaged parallel execution times for frequency decomposition and section decomposition, respectively. Each execution time measured is the effective execution time, which represents the amount of time between successive outputs.



a) PC cluster



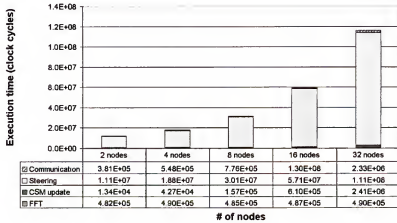
b) DSP array

Figure 5.9. Parallel execution time of frequency decomposition vs. number of nodes

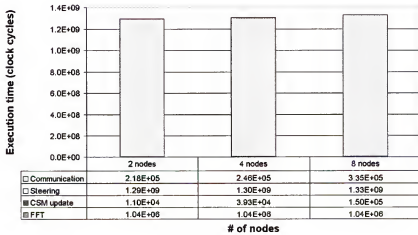
The execution time of the FFT stage does not change significantly as the number of nodes increases due to inherent parallelism from the architecture, as mentioned earlier.



This linear scaling model for the processing unit in the parallel algorithms decreases the degree of the complexity polynomial by one. For instance, the FFT complexity in this implementation will be  $O(1)$  from  $O(N)$ . The additional workload caused by increasing the number of sensor nodes is therefore evenly distributed across the processors in this case. As a result, the number of sensor nodes does not influence the execution time of the stage. For other stages, reduced complexity of the stage can be observed in the execution times of both parallel algorithms as well.



a) PC cluster



b) DSP array

Figure 5.10. Parallel execution time of section decomposition vs. number of nodes

On the PC cluster, the two parallel algorithms show a substantial discrepancy in execution time for communication and CSM update. As explained in Section 5.3, the frequency decomposition has to deal with a larger message size than section decomposition, and the all-to-all communication time increases rapidly with increased problem size. This collective communication is sensitive to the message size in terms of time. The dependency between the CSM update and steering stage prevents parallel implementation of the CSM update stage in section decomposition. The increase in CSM update time in the section decomposition originates from this sequential bottleneck.

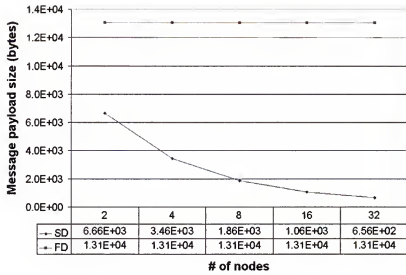
On the DSP array, both parallel algorithms achieve about the same execution times. The major portion of the execution time comes from the steering time, which includes steering vector generation. Steering itself takes quite a small portion of the time compared to the generation, which shows significant execution time. By parallel implementation, the reduced complexity for steering and steering vector generation is  $O(N)$  and  $O(1)$ , respectively. For the given problem size, the almost constant execution time for the steering vector generation stage obscures linear growth of execution time caused by the steering itself. The complexity of communication is somewhat high in both parallel algorithms because of the all-to-all pattern. However, communication time increases slowly for a given problem size since the DSP array efficiently handles the collective communication. Additional information about communication can be found in the next subsection.

### 5.5.3. Communication Time

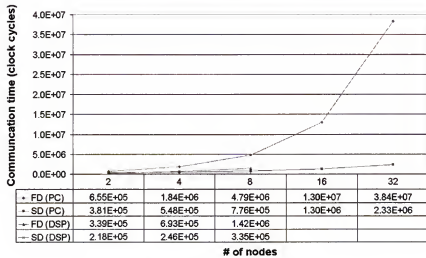
The parallel execution time consists of two important factors, computation time and communication time. The computation time generally relies on the decomposition method, and the communication time is determined by several issues, such as granularity,

message size, communication method, and so forth. As described in the previous section, the message size between the two parallel algorithms is a major difference. The computed message size per iteration is shown in Figure 5.11a, and measured communication times for the two testbeds and two parallel algorithms are presented in Figure 5.11b. The frequency decomposition shows a constant message size versus system size and a rapid communication time increase on the PC cluster. In contrast, the section decomposition has a decreasing message size and moderate communication time growth on the PC cluster.

Since the MPI-SHARC design on the DSP array implements a form of hardware broadcast, the bandwidth achieved is comparable to high-performance network architectures [KOH01]. The communication measurements from the DSP array reflect this design consideration. The time increase for communication is shown as a gentle slope for both parallel algorithms on the DSP array. Although the frequency decomposition requires more communication time than section decomposition on the DSP array, the difference is modest and both outperform their counterpart on the PC cluster.



a) message size per node



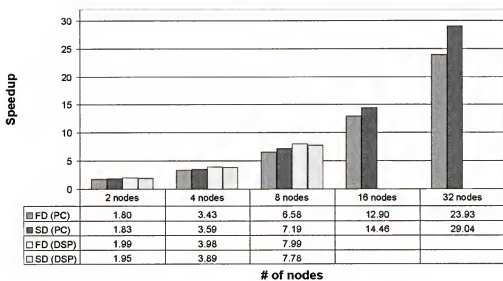
b) communication time

Figure 5.11. Communication with parallel algorithms

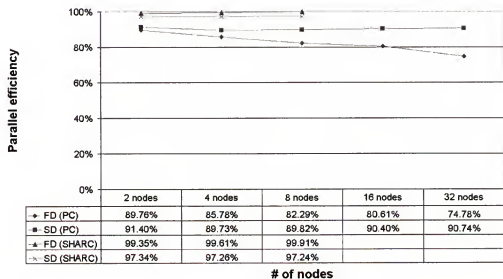
#### 5.5.4. Scaled Speedup and Parallel Efficiency

Figure 5.12 shows the overall system performance in terms of scaled speedup and parallel efficiency. The baseline for comparison is the sequential CMFP algorithm running on one PC or DSP of the same testbed. Since the algorithms incur additional

workload as sensors are added, Figure 5.12a shows scaled speedup. Figure 5.12b displays scaled parallel efficiency, which is defined as scaled speedup divided by the ideal speedup, the number of processors used.



a) scaled speedup



b) parallel efficiency

Figure 5.12. Parallel performance

On speedup, section decomposition shows better performance because of less communication overhead on the PC cluster. However, on the DSP array, frequency decomposition provides better performance than section decomposition due to the well-balanced workload and efficient communication of the DSP array. Thus, the parallel algorithms on the PC cluster are sensitive to the communication method because of the inefficient collective communication. By contrast, communication is managed well on the DSP array so the performance dependency from the communication is subtle. Rather, balanced workload plays an important role on the performance on the DSP array. Between the two testbeds, parallel algorithms on the DSP array achieve a more scalable performance because of the model selection and efficient communication on the DSP array.

Significant communication overhead from the PC cluster produces performance degradation on frequency decomposition, as shown in the form of decreasing parallel efficiency in Figure 5.12b. For section decomposition on the PC cluster, the communication overhead and parallel overhead are not significant; hence, almost constant parallel efficiency is achieved. Both parallel algorithms on the DSP array achieve almost ideal scalability for a given problem size.

#### 5.5.5. Data Memory Capacity

In Section 5.2, we chose the MC model for the PC cluster and the MM model for the DSP array as the baseline for this investigation. In both models, the majority of the memory requirement arises from the steering stage. Due to the model selection, the sequential algorithm on the PC cluster requires an extensive amount of memory for steering vectors, but on the DSP array it consumes a small constant memory for steering, as shown in Figure 5.13. Moreover, as compared to the sequential baseline, both parallel

algorithms need only a fraction of the memory space for steering per node since individual processors generate only part of the result for a given data set. Therefore, it is necessary that each processor from the parallel algorithms keeps part of the steering vectors and parameters for partial results. Frequency decomposition requires the least memory and shows a decreasing trend since data can be further divided across multiple nodes, since there is no dependency on basic parameters, such as eigenvalues and eigenvectors, between frequency bins. The parallel algorithms introduced in this paper distribute memory requirements efficiently; hence, these algorithms can be used on systems with memory-constrained nodes.

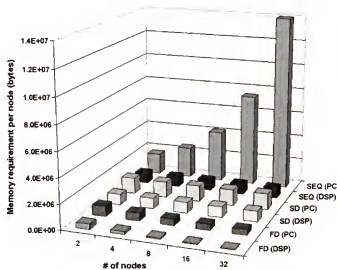


Figure 5.13. Data memory requirement per node for the steering stage

### 5.5.6. Power and Energy Consumption

Power and energy consumption are becoming a critical design issue in contemporary systems. For mobile and/or deployable systems, power and energy are important constraints in the design specification because of battery capacity. Since the

final destination of these parallel algorithms may be battery-operated sonar array systems, such as portable or rapidly deployable systems, the power and energy parameters are considered with a hardware experiment to measure the instruction-level power consumption of the ADSP-21062. In this experiment, a low-resistance resistor is inserted into the major power line of the DSP board. The voltage drop over the resistor is proportional to the current bypassing through the power line. We can derive the power consumption of the board by measuring the voltage. The voltage is monitored by the oscilloscope to capture the instruction-level power consumption, and the results are measured many times and averaged for reliable results.

Table 5.1. Type of instruction and corresponding power consumption

Type	Symbol	Instruction	Power consumption (watts)
<i>Idle Operation</i>	<i>D</i>	Idle	1.77
<i>No Operation</i>	<i>N</i>	Nop	2.35
<i>Single Operation</i>	<i>S</i>	Assignment	2.50
		Integer Add	
		Integer Subtract	
		Float Add	
		Float Subtract	
		Integer Multiplication	
		Float Multiplication	
		DAG register assignment	
		Jump	
		Compare	
<i>Internal Memory Operation</i>	<i>I</i>	Internal Memory Read	2.67
		Internal Memory Write	
<i>External Memory Operation</i>	<i>E</i>	External Memory Read	5.03
		External Memory Write	



Since the DSP has many types of instructions, it is useful to group instructions for power measurement. The categorized instructions are applied to the testbed for power measurement. The results are shown in Table 5.1. In this experiment, we assume that there is no correlation and dependency on power consumption between the different types of instruction. From this assumption, the total consumed power can be calculated by the linear combination of individual power consumption of each instruction. More extensive work on instruction-level power measurement of microprocessors can be found in Tiwari et al. [TIW94].

Access to an external unit, such as an external memory operation, requires substantially more power than an internal type of instruction. Within the processor, the internal memory is in the form of an extra unit; therefore, the internal memory operation presents a slightly higher consumption than single internal instructions. The idle instruction places the DSP in the lowest power state in the operating mode because the processor remains in the halt state until an external interruption occurs to awaken the processor. Based on the given information in Table 5.1, the total power consumption is estimated by Equation 5.6. In the equation, each symbol is the corresponding instruction frequency, which is the ratio between the number of the corresponding instructions and the total instruction count per iteration. This ratio indicates the fraction of occurrences of the corresponding instruction for a single iteration as measured from the instruction mix of the program under test. Note that the sum of all symbol values,  $D+N+S+I+E$ , always indicates one. The computed results are shown in Figure 5.14.

$$\begin{aligned} \text{Average power consumption (watts)} = & D \times 1.77\text{W} + N \times 2.35\text{W} + S \times 2.50\text{W} + \\ & I \times 2.67\text{W} + E \times 5.03\text{W} \end{aligned} \quad (5.6)$$

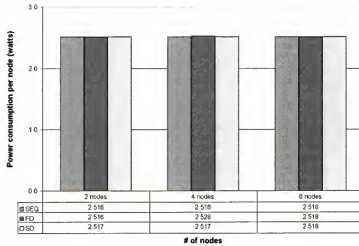
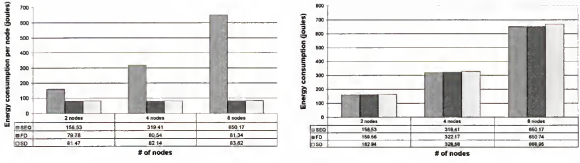


Figure 5.14. Power consumption of CMFP in the DSP array

Since the sequential model for the DSP array is MM, the three algorithms shown in Figure 5.14 seldom use external memory. Most of the instructions executed on the DSP array are single operations in these cases. Without external memory access, the sequential and parallel algorithms incur almost identical power consumption, which is similar to the single-operation power consumption.

Since the power is a time-averaged parameter, there is no information about time. To consider the power and time simultaneously, the energy parameter is introduced. The unit of the energy is a joule, and one joule is the amount of energy needed to lift one pound about nine inches. One watt is equal to one joule per second so the basic energy equation is given as follows.

$$\text{Energy (joules)} = \text{Power (watts)} \times \text{Time (seconds)} \quad (5.7)$$



a) energy consumption per node

b) energy consumption for the whole system

Figure 5.15. Energy consumption of CMFP in the DSP array

Figure 5.15 shows energy consumption per iteration for each processor and the whole system, respectively. As expected, each individual processor used with the parallel algorithm uses much less energy per iteration than a single processor with the sequential algorithm, which means that the parallel algorithms perform the same amount of work as sequential algorithm with a smaller capacity of battery per processor. The total energy consumption from three algorithms is almost identical, as seen in Figure 5.15b. The parallel algorithms show slightly increased total energy consumption versus than sequential algorithm because of overhead for parallel implementation. Consequently, the total energy required is distributed across the processors with the parallel algorithms and thus they do not require a single massive battery for operation as does the sequential baseline. Therefore, each sensor node is operated with a small battery for the system's physical balance. From this perspective, an energy distribution efficiency is formulated below.

$$\text{Energy distribution efficiency} = \frac{\text{Sequential energy requirement}}{\text{Parallel energy requirement per processor} \times \# \text{ of processors}} \quad (5.8)$$

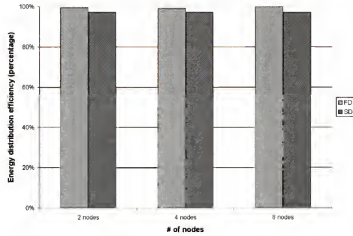


Figure 5.16. Energy distribution efficiency of parallel CMFP

The energy distribution efficiency illustrates that the implementation is able to evenly distribute the energy requirement over the multiple processors in the system. For example, a perfectly balanced distribution computes to 100% from this formula. Both parallel algorithms show nearly ideal energy distribution in Figure 5.16.

### 5.6. Summary

The parallel algorithms introduced in this paper distribute the task of CMFP in two different domains: frequency and output points. By using a data packing method, these parallel algorithms reduce communication time and granularity of the algorithm to improve the parallel performance over the distributed system. Two distinct parallel testbeds were employed to study the parallel performance in terms of several parameters including scalable speedup, parallel efficiency, memory requirements, power/energy consumption, and so forth.

Overall, both parallel algorithms show promising performance. Except for frequency decomposition on the PC cluster, parallel efficiency results achieve more than

90% up to a system size of 32 nodes for the PC cluster and 8 nodes for the DSP array. Frequency decomposition on the PC cluster shows a rapidly decreasing parallel efficiency from 90% for 2 nodes to 75% for 32 nodes because of significant communication overhead. By contrast, the advantage of short messages on section decomposition makes the parallel efficiency stay nearly flat.

The PC cluster consists of fast processing units with relatively inefficient communication channels. In contrast, the DSP array is built with slower processing units with efficient communication channels. Therefore, the performance of parallel algorithms is susceptible to these inefficient or time-consuming parts of the testbed. The results from this investigation reveal these tradeoffs. For example, performance degradation is observed with frequency decomposition on the PC cluster and with section decomposition on the DSP array. Of the two testbeds, the DSP array demonstrates better relative performance with the two parallel algorithms due to the model selection and communication method. Both parallel algorithms on both testbeds require significantly less memory per node for the steering stage due to domain decomposition. Finally, the results show that the energy requirement of the sequential algorithm is almost ideally distributed over the multiple processors with the parallel algorithms.

## CHAPTER 6 CONCLUSIONS

In this dissertation, the performance limitations of several prominent sequential beamforming algorithms have been studied. Parallel beamforming algorithms have been introduced as a solution that increases the overall computational performance, dependability, and versatility of these algorithms for sonar array systems.

The parallel algorithms implemented herein distribute sequential workload based on several perspectives including time, angle, frequency, and output points. A range of communication schemes has been employed to reduce the overhead caused from parallel implementation as well. Data packing and overlapped execution by pipelining are able to relieve the communication overhead to a certain extent. The relationship between computation and communication is analyzed by using the results from parallel processing experiments. Three distinct experimental testbeds are used in this dissertation, including a cluster of Unix workstations, a cluster of Linux PCs, and an array of DSPs. In addition to speed and memory issues, power and energy consumption are also considered in the end in support of battery-powered distributed and deployable systems.

In the first part of this dissertation, parallel algorithms for SA-CBF were designed to improve the performance in terms of execution speed and the memory. Due to the least amount of interprocessor communication, the coarse-grained algorithm, known as iteration decomposition, is able to improve the beamforming throughput at the expense of an increased memory requirement. Iteration decomposition is able to achieve almost 90% in parallel efficiency. However, parallel efficiency decreases by almost 35% from a

2-node to an 8-node system with angle decomposition because of its dominant all-to-all communication overhead relative to computation complexity.

In the second part of this dissertation, parallel algorithms for iteration decomposition and frequency decomposition were devised for SPB, a computationally intensive adaptive beamforming algorithm. Overall, both of the parallel algorithms achieve scalable parallel performance with increasing parallel efficiency. Due to the high computational complexity of the SPB algorithm, the overhead caused by parallel coordination and communication is concealed by dominant computation times for larger system sizes. For example, on the Linux PC cluster, it was observed that parallel efficiency increased by almost 22% and 35% from a 2-node to a 32-node system with iteration decomposition and frequency decomposition, respectively. The parallel efficiency of the iteration decomposition is bounded at the lower problem sizes by the pipeline overhead and imbalance of the parallel algorithm rather than by interprocessor communication. Of the two algorithms, frequency decomposition is the better overall choice for the problem sizes considered, since it shows better scalable performance with a smaller memory requirement and shorter result latency as a ratio of system size. However, iteration decomposition is promising when employed on systems with low communication subsystem performance or when the problem size grows such that communication complexity begins to be dominant.

Finally, complex-wave beamforming with CMFP is considered for parallel algorithm design and implementation in the final phase of the dissertation. Two parallel algorithms are implemented on two testbeds for the performance study. Both parallel algorithms, frequency decomposition and section decomposition, exhibit all-to-all

communication and therefore the more efficient communication channel on the DSP array is found to be a major advantage in gaining better relative performance than on the PC cluster. Except for frequency decomposition on the PC cluster, all of the results show more than 90% parallel efficiency up to a system size of 32 nodes for the PC cluster and up to a system size of 8 nodes for the DSP array.

This dissertation is aimed at gaining important insights into the design and performance characteristics of parallel beamforming algorithms. This objective requires interdisciplinary research in the areas of computer engineering and digital signal processing. Novel parallel algorithms, coupled with high-performance computers and networks, are leveraged with one another to allow the parallel system to overcome the computational challenges of beamforming algorithms so that advanced beamforming algorithms can be implemented within the real-time constraints of sonar processing systems.

The parallel beamforming techniques described in this paper present many opportunities for increased performance, reliability, and flexibility in a distributed parallel sonar array. These parallel methods provide considerable speedup with multiple sensor nodes, thus enabling previously unfeasible algorithms to be implemented in real-time. Furthermore, the fault tolerance of the sonar architecture can potentially be increased by taking advantage of the distributed nature of these parallel algorithms and avoiding single points of failure. Future work will involve several new avenues of research for distributed and parallel sonar processing. The distributed and parallel techniques developed herein can be extended to more advanced forms of sonar signal processing such as adaptive MFP algorithms. High-fidelity power and energy models



need to be constructed and verified for distributed DSP arrays. Based on such models, next-generation sonar array systems may be able to use power more efficiently by employing smart power management algorithms to extend the mission time.

## LIST OF REFERENCES

- [AMD67] G. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," Proc. AFIPS 1967 Spring Joint Conf., Vol. 30, pp. 483-485 (1967).
- [BAG93] A. B. Baggeroer, W. A. Kuperman, and P. N. Mikhalevsky, "An overview of matched field methods in ocean acoustics," IEEE J. Ocean Eng. Vol. 18, No. 4, pp. 401-423 (1993).
- [BAR48] M. S. Bartlett, "Smoothing periodograms from time series with continuous spectra," Nature, Vol. 161, pp. 686-687 (1948).
- [BER96] R. Bernecky, "Sonar beamforming challenge problems," DARPA/ITO Embeddable Systems PI Meeting, San Diego, CA (1996).
- [BIT97] Bittware Research Systems, User's Guide: Blacktip-EX, Bittware Research Systems, Concord, NH, Revision 0 (1997).
- [BUC76] H. P. Bucker, "Use of calculated sound fields and matched-field detection to locate sound sources in shallow water," J. Acoust. Soc. Am., Vol. 59, No. 2, pp. 368-373 (1976).
- [CAS95] L. Castedo and A. R. Figueiras-Vidal, "An adaptive beamforming technique based on cyclostationary signal properties," IEEE Trans. Signal Processing, Vol. 43, No. 7, pp. 1637-1650 (1995).
- [CUL94] C. G. Cullen, "An introduction to numerical linear algebra," PWS Publishing Company, Boston (1994).
- [DAV67] D. E. N. Davies, "Independent angular steering of each zero of the directional pattern for a linear array," IEEE Trans. Antennas and Propagation, Vol. AP-15, pp. 296-298 (1967).
- [FER89] B. G. Ferguson, "Improved time-delay estimates of underwater acoustic signals using beamforming and prefiltering techniques," IEEE J. Ocean Eng. Vol. 14, No. 3, pp. 238-244 (1989).
- [FIZ85] R. G. Fizell and S. C. Wales, "Source localization range and depth in an arctic environment," J. Acoust. Soc. Am., Vol. 78, p. S5 (1985).

- [GEO98] A. D. George, R. Fogarty, J. Garcia, K. Kim, J. Markwell, M. Miars, and S. Walker, "Parallel and distributed computing architectures and algorithms for fault-tolerant sonar arrays," Annual Report, prepared for the Office of Naval Research, Arlington, Virginia (1998).
- [GEO99] A. George and K. Kim, "Parallel algorithms for split-aperture conventional beamforming," *Journal of Computational Acoustics*, Vol. 7, No. 4, pp. 225-244 (1999).
- [GEO00] A. George, J. Markwell, and R. Fogarty, "Real-time sonar beamforming on high-performance distributed computers," *Parallel Computing*, Vol. 26, No. 10, pp. 1231-1252 (2000).
- [GEO01] A. George, J. Garcia, K. Kim, and P. Sinha, "Distributed parallel processing techniques for adaptive sonar beamforming," *Journal of Computational Acoustics*, accepted and in press.
- [GIN93] D. F. Gingras, "Robust broadband matched-field processing: performance in shallow water," *IEEE J. Ocean Eng.* Vol. 18, No. 3, pp. 253-264 (1993).
- [GOL96] G. H. Golub and C. F. Van Loan, "Matrix computations, 3rd edition," The Johns Hopkins University Press, Baltimore and London (1996).
- [GRO96] W. Gropp, E. Lusk, N. Doss and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," *Parallel Computing*, Vol. 22, No. 6, pp. 789-828 (1996).
- [GUS88] J. L. Gustafson, "Reevaluating Amdahl's law," *Comm. ACM*, Vol. 31, No. 5, pp. 532-533 (1998).
- [HAG88] W. W. Hager, "Applied numerical linear algebra," Prentice-Hall, Englewood Cliffs, NJ (1988).
- [HSI96] S. Hsiao and J. Delosme, "Parallel singular value decomposition of complex matrices using multidimensional CORDIC algorithms," *IEEE Trans. Signal Processing*, Vol. 44, No. 3, pp. 685-697 (1996).
- [HWA93] K. Hwang, "Advanced computer architecture: parallelism, scalability, programmability," McGraw-Hill, New York (1993).
- [JEN94] F. B. Jensen, W. A. Kuperman, M. B. Porter, and H. Schmidt, "Computational ocean acoustics," AIP Press, New York (1994).
- [KOH01] J. Kohout, "Design and performance analysis of MPI-SHARC: a high-speed network service for distributed DSP systems," Master's thesis, ECE Department, University of Florida (Spring 2001).

- [KRI96] H. Krim and M. Viberg, "Two decades of array signal processing research," IEEE Signal Processing Magazine, pp. 67-94 (July 1996).
- [KRO89] J. Krolik and D. Swingler, "Multiple broad-band source location using steered covariance matrices," IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. 37, No. 10, pp. 1481-1494 (1989).
- [LIU98] W. Liu and V. Prasanna, "Utilizing the power of high-performance computing," IEEE Signal Processing Magazine, pp. 85-100 (Sept. 1998).
- [MAC90] F. Machell, "Algorithms for broadband processing and display," ARL Technical Letter No. 90-8 (ARL-TL-EV-90-8), Applied Research Laboratories, Univ. of Texas at Austin (1990).
- [MAR68] R. S. Martin and J. H. Wilkinson, "The implicit QL algorithm," Numer. Math., Vol. 12, pp. 377-383 (1968).
- [MPI94] Message Passing Interface Forum, "MPI: a message-passing interface standard," Technical Report CS-94-230, Computer Science Dept., Univ. of Tennessee (Apr. 1994).
- [NUT81] Albert H. Nuttall, "Some windows with very good sidelobe behavior," IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. ASSP-29, No. 1, pp. 84-91 (1981).
- [PAC95] P. Pacheco, "A user's guide to MPI," Department of Mathematics, University of San Francisco (1995).
- [POR84] M. Porter and E. L. Reiss, "A numerical method for ocean-acoustic normal modes," J. Acoust. Soc. Am., Vol. 76, No. 1, p. 244 (1984).
- [POR91] M. Porter, The KRAKEN normal mode program, SCALANT Memorandum, SM-245 (Sept. 1991).
- [POR93] M. B. Porter, "Acoustic models and sonar systems," IEEE J. Ocean Eng. Vol. 18, No. 4, pp. 425-437 (1993).
- [POR94] M. Porter, Kraken Normal mode program,  
<ftp://oalib.saic.com/pub/oalib/demo/modes.m>
- [POR97] M. Porter, "Ocean acoustic modeling in MATLAB," Center for Applied Mathematics and Statistics, New Jersey Institute of Technology,  
<ftp://oalib.saic.com/pub/oalib/demo/MatMod.ps> (1997).

- [ROB91] W. Robertson and W. J. Phillips, "A system of systolic modules for the MUSIC algorithm," IEEE Trans. Signal Processing, Vol. 39, No. 11, pp. 2524-2534 (1991).
- [SCH81] R. Schmidt, "A signal subspace approach to multiple emitter location and spectral estimation," Ph.D. dissertation, Stanford Univ. (1981).
- [SMI96] M. J. Smith and I. K. Proudler, "A one sided algorithm for subspace projection beamforming," Proc. SPIE: Advanced signal processing algorithms, architectures and implementations VI, pp. 100-111, Denver, Colorado (Aug. 4-9, 1996).
- [SNI96] M. Snir, S. Huss-Lederman, D. Walker, and S. W. Otto, "MPI: The complete reference," MIT Press, Cambridge (1996).
- [STE89] A. O. Steinhardt and B. D. Van Veen, "Adaptive beamforming," International J. of Adaptive Control and Signal Processing, Vol. 3, pp. 253-281 (1989).
- [STE91] S. Stergiopoulos and A. T. Ashley, "An experimental evaluation of split-beam processing as a broadband bearing estimator for line array sonar systems," J. Acoust. Soc. Am., Vol. 102, No. 6, pp. 3556-3563 (1991).
- [STE95] T. Sterling, D. Becker, D. Savarese, et al. "BEOWULF: A Parallel Workstation for Scientific Computation," Proceedings of the 1995 International Conference on Parallel Processing (ICPP), Vol. 1, pp. 11-14, (Aug. 1995).
- [STO89] P. Stoica and A. Nehorai, "MUSIC, maximum likelihood, and Cramer-Rao Bound," IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. 37, No. 5, pp. 720-741 (1989).
- [TIW94] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization," IEEE Transactions on VLSI Systems, Vol. 2, No. 4 (1994).
- [UNI69] United States, Office of Scientific Research and Development, National Defense Research Committee, "Physics of sound in the sea," Dept. of the Navy, Headquarters Naval Material Command, Washington, DC, pp. 8-40 (1969).
- [URI75] R. J. Urick, "Principles of underwater sound," McGraw-Hill, New York (1975).
- [VAC98] R. J. Vaccaro, "The past, present, and future of underwater acoustic signal processing," IEEE Signal Processing Magazine, pp. 21-51 (1998).

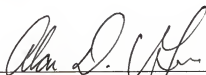
- [ZHA97] M. Zhang and M. H. Er, "Robust adaptive beamforming for broadband arrays," *Circuits, Systems, and Signal Processing*, Vol. 16, No. 2, pp. 207-216 (1997).
- [ZOM96] A. Zomaya, "Parallel and distributed computing handbook," McGraw-Hill, New York (1996).
- [ZUI93] N. V. Zuikova and V. D. Svet, "Matched-field processing of signals in ocean waveguides (review)," *Sov. Phys. Acoust.*, Vol. 39, No. 3, pp. 203-210 (1993).

## BIOGRAPHICAL SKETCH

Keonwook Kim was born in Pusan, Korea, in 1969. After graduating from Pusan Jung-Ang High School in 1988, he attended the Dongguk University, Seoul, where he was awarded a Bachelor of Science in Electronics Engineering in 1995. Keonwook entered the graduate program for Master of Science and Doctor of Philosophy at the University of Florida in August of 1995 and joined the *High-performance Computing and Simulation (HCS) Research Laboratory* in May of 1997 as a doctoral student. He served as team leader for the High-performance Applications (HPA) team beginning in the Spring of 1999. He has been involved in many projects in the doctoral program, including high-performance computing, distributed and parallel processing, beamforming algorithms, cluster computing, and computer architecture.

Keonwook's research was supported by the Office of Naval Research. After receiving his Ph.D., Keonwook will be taking a faculty position as an assistant professor of electrical and computer engineering at Florida State University and Florida A&M University in the FAMU-FSU College of Engineering, Tallahassee, FL.

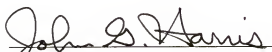
I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



---

Alan D. George, Chairman  
Associate Professor of Electrical and  
Computer Engineering

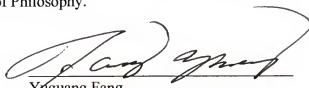
I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



---

John G. Harris  
Associate Professor of Electrical and  
Computer Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

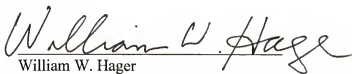


---

Yuguang Fang  
Assistant Professor of Electrical and  
Computer Engineering



I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

  
William W. Hager  
Professor of Mathematics

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

August 2001

  
Pramod P. Khargonekar  
Dean, College of Engineering

\_\_\_\_\_  
Winfred M. Phillips  
Dean, Graduate School